IMPLEMENTASI ALGORITMA BLIND WATERMARKING MENGGUNAKAN METODE FRACTIONAL FOURIER TRANSFORM DAN VISUAL CRYPTOGRAPHY

Nama : Mir'atul Mahmudah
NRP : 5110 100 131
Jurusan : Teknik Informatika – FTIf ITS
Dosen Pembimbing I : Dr. Eng. Nanik Suciati, S.Kom., M.Kom.
Dosen Pembimbing II : Arya Yudhi Wijaya, S.Kom., M.Kom.

Abstrak

Digital image watermarking merupakan sebuah teknik yang digunakan untuk menyembunyikan informasi ke dalam citra digital. Algoritma watermarking yang dapat mengekstraksi citra watermark tanpa menggunakan informasi citra host disebut blind watermarking. Skema ini dinilai lebih efektif dan efisien, karena dapat menjaga keberadaan citra host asli. Pada tugas akhir ini penulis mengusulkan implementasi algoritma blind watermarking menggunakan metode fractional Fourier transform dan visual cryptography untuk memperbaiki kelemahan algoritma blind watermarking konvensional.

Tidak seperti watermarking konvensional, algoritma yang diusulkan dapat melakukan watermarking tanpa harus memodifikasi citra host, sehingga kualitas citra host tetap tinggi. Visual cryptography degan konsep visual secret sharing digunakan untuk membangun dua bagian dari citra watermark, yaitu master share dan ownership share. Master share dibangun dari fitur citra host yang didapatkan dengan melakukan transformasi citra dan singular value decomposition. Sedangkan ownership share dibangun dari master share dan citra watermark menggunakan teknik visual cryptography. Ownership share harus didaftarkan kepemilikannnya untuk keamanan lebih lanjut. Proses identifikasi citra watermark pada skema ini dapat dilakukan dengan cara

menumpuk master share dari citra yang diklaim dan ownership share yang sudah disimpan.

Hasil uji coba menunjukkan bahwa ownership share yang dihasilkan hanya dapat digunakan untuk mengektraksi citra yang bersangkutan. Citra watermark yang disisipkan juga memiliki ketahanan yang baik terhadap berbagai kemungkinan gangguan, seperti blurring, sharpening, kompresi JPEG, resizing, penambahan derau, rotasi dan cropping.

Kata kunci: Blind watermarking, fractional Fourier transform, singular value decomposition, visual secret sharing

IMPLEMENTATION OF BLIND WATERMARKING ALGORITHM USING FRACTIONAL FOURIER TRANSFORM AND VISUAL CRYPTOGRAPHY

Name	: Mir'atul Mahmudah
NRP	: 5110 100 131
Department	: Informatics Engineering – FTIf ITS
Advisor I	: Dr. Eng. Nanik Suciati, S.Kom., M.Kom.
Advisor II	: Arya Yudhi Wijaya, S.Kom., M.Kom.

Abstract

Digital image watermarking is technique wich can used to hide information. Watermarking scheme wich can extract the watermark image without using the host image is called blind watermarking. This scheme is more effective and efficient, because it can maintain the existence of the original host image. In this final project the author proposes implementation of blind watermarking algorithm using fractional Fourier transform and the visual cryptography to rectify the shortcomings of conventional blind watermarking.

Unlike the conventional watermarking, the proposed algorithm can perform watermarking without the host image, so the quality of host image is high. Visual cryptography with visual secret sharing scheme is used to generate two different shares, namely the master share and the ownership share. Features of the original image are extracted by performing image transformation and singular value decomposition, and are used to generate the master share.The ownership share is generated with help of master share and watermark images using visual cryptography technique. Ownership share is registered at some certification authority which can be used for checking authenticity of image or digitized data in case of any attack.The process of identification of the watermark image on this scheme can be reveal by stacking the master share of the watermarked image and the ownership share.

The experimental results show that the ownership share only can be used to extract watermark from the right host image. The watermark image has also good robustness from various attacks, such as blurring, sharpening, JPEG compression, noise addition, rotation and cropping.

Keywords: Blind watermarking, fractional Fourier transform, singular value decomposition, visual secret sharing

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan algoritma yang diajukan pada pengimplementasian perangkat lunak. Penjelasan ini bertujuan untuk memberikan gambaran umum terhadap aplikasi yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1. *Watermarking* Citra

Watermarking citra adalah suatu cara menyembunyikan atau menyisipkan data baik berupa teks maupun gambar pada suatu media citra digital [1]. *Watermarking* citra diimplementasikan dengan memanfaatkan keterbatasan sistem indera manusia khususnya mata, yaitu dengan menurunkan kualitas warna pada citra yang belum disisipi *watermark*. Dengan adanya keterbatasan inilah, metode *watermarking* dapat diterapkan pada berbagai media citra digital. Hasil keluaran dari *watermarking* memiliki bentuk persepsi yang sama dengan bentuk aslinya, tentunya persepsi ini sebatas oleh kemampuan indera manusia, tetapi tidak oleh komputer atau perangkat pengolah digital lainnya.

Berdasarkan metode pemrosesnya, *watermarking* dapat digolongkan menjadi dua bagian yaitu, pada domain spasial dan domain transformasi. *Watermarking* yang bekerja pada domain spasial langsung mengubah nilai piksel pada citra asli. Metode tersebut memiliki kompleksitas komputasi yang rendah namun tidak tahan terhadap serangan. Sebaliknya teknik *watermarking* dalam domain transformasi memiliki lebih banyak keuntungan dan kinerja yang lebih baik daripada teknik yang bekerja pada domain spasial. Berbagai transformasi yang biasanya digunakan dalam pemrosesan sinyal digital, diantaranya yaitu FFT (*Fast Fourier Transform*), DCT (*Discrete Cosine Transform*), DWT (*Discrete Wavelet Transform*).

Penggunaan *watermarking* citra salah satunya bertujuan untuk melakukan perlindungan hak cipta citra tersebut. Berkaitan dengan hal ini maka diperlukan suatu algoritma *watermarking* yang memiliki kinerja yang baik. Kinerja sebuah algoritma *watermarking* yang baik dapat dinilai dari beerapa faktor, yaitu;

1. *Imperceptibility*: keberadaan pesan rahasia dalam media penampung tidak dapat dideteksi oleh inderawi.
2. *Fidelity*: mutu media penampung tidak berubah banyak akibat penyisipan. Perubahan itu tidak dapat dipersepsi oleh inderawi.
3. *Recovery*: pesan yang disembunyikan harus dapat diungkapkan kembali (*reveal*). Dikarenakan tujuan *watermarking* adalah *data hiding*, maka sewaktu-waktu pesan rahasia di dalam citra ter-*watermarked* harus dapat diambil kembali untuk digunakan lebih lanjut.
4. *Security*: Tidak seorang pun dapat mengesktraksi atau menghapus *watermark* yang disisipkan kecuali pemilik sah dari media digital tersebut.
5. *Blindness*: Citra *host* asli tidak dibutuhkan di dalam proses ekstraksi *watermark*.
6. *Robustness*: pesan yang disembunyikan harus tahan (*robust*) terhadap berbagai operasi manipulasi atau *editing* pada citra ber-*watermark*. Ini berarti manipulasi yang dilakukan terhadap citra ber-*watermark* tidak merusak *watermark* (*watermark* masih dapat dideteksi). Manipulasi citra meliputi kompresi JPEG, penambahan derau, *sharpening*, *blurring*, *resizing*, *rotation* maupun *cropping*.

Terdapat dua proses utama dalam *watermarking* citra, yaitu proses penyisipan (*embedding process*) dan proses ekstraksi (*extraction process*). Proses penyisipan adalah proses di mana *watermark* disisipkan ke dalam citra *host*. Proses ini dapat disertai dengan pemasukan kunci atau tidak memerlukan kunci. Kunci diperlukan agar hanya dapat diekstraksi oleh pihak yang sah. Kunci juga bermanfaat untuk mencegah *watermark* dihapus oleh pihak yang tidak berhak. Sedangkan ketahanan terhadap proses-proses

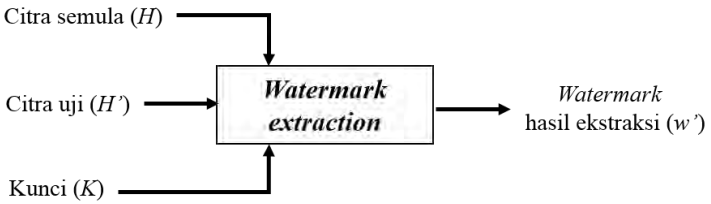
pengolahan lainnya tergantung pada metode *watermarking* yang digunakan.

Proses ekstraksi adalah proses mengekstrak kembali *watermark* dari citra yang dihasilkan dari proses penyisipan sebelumnya. Ekstraksi *watermark* dilakukan untuk membuktikan status kepemilikan citra digital yang disengketakan. *Watermark* harus dapat diekstraksi atau dideteksi kembali bergantung pada sifat dan tujuan algoritma *watermarking*. Pada beberapa algoritma *watermarking*, *watermark* dapat diekstraksi dalam bentuk yang eksak, sedangkan pada sebagian algoritma yang lain hanya dapat dideteksi apakah *watermark* terdapat di dalam citra, sehingga prosedurnya dinamakan pendeteksian *watermark*.

Berdasarkan cara ekstraksi *watermark*-nya, *watermarking* dapat dibedakan menjadi *blind watermarking* dan *non blind watermarking* [1]. Kedua jenis algoritma *watermarking* ini dijelaskan lebih lanjut pada subbab berikut.

2.1.1. *Non Blind Watermarking*

Algoritma *watermarking* yang mengikutsertakan citra *host* asli pada saat ekstraksi disebut *non blind watermarking*. Algoritma jenis ini dinilai tidak efektif, karena dalam proses *watermarking* kita dituntut untuk menjaga citra asli. Namun, penggunaan citra *host* asli pada proses ekstraksi dapat meningkatkan hasil ekstraksi *watermark* yang lebih baik. Proses ekstraksi *watermark* pada algoritma ini ditunjukkan pada Gambar 2.1.

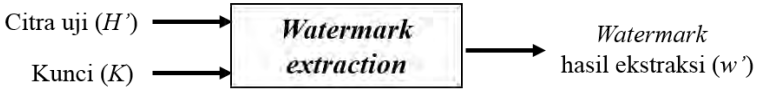


Gambar 2.1 Proses Ekstraksi pada Algoritma *Non Blind Watermarking*

2.1.2. *Blind Watermarking*

Algoritma *watermarking* yang tidak membutuhkan citra *host* asli pada saat ekstraksi disebut *blind watermarking*. Proses ekstraksi *watermark* pada algoritma ini ditunjukkan pada Gambar 2.2. Algoritma *blind watermarking* lebih banyak digunakan karena lebih efektif. Namun beberapa penelitian yang menerapakan algoritma ini, membutuhkan proses ekstraksi yang cukup rumit untuk mendapatkan hasil ekstraksi yang baik.

Pada tugas akhir ini dikembangkan pendekatan algoritma *blind watermarking* dengan menggunakan konsep VSS yang dapat melakukan ekstraksi *watermark* dengan cara yang lebih sederhana. Penjelasan mengenai konsep VSS dijelaskan lebih lanjut pada subbab berikutnya.



Gambar 2.2 Proses Ekstraksi pada Algoritma *Blind Watermarking*

2.2. Penelitian Sebelumnya

Penelitian tentang *watermarking* sudah banyak diterapkan dengan berbagai macam metode. Wang [5] mengusulkan *watermarking* dengan teknik *visual cryptography* pada domain spasial dengan menyembunyikan *watermark* pada blok tepi citra *host*. Pada skema ini, *watermark* dibagi menjadi *watermark* publik dan *watermark* rahasia dengan menggunakan skema VSS untuk meningkatkan keamanan teknik *watermarking* yang diusulkan. Berbeda dengan metode tradisional, *watermark* asli tidak harus tertanam ke dalam citra *host* secara langsung dan dengan demikian sulit untuk dideteksi atau dihapus *hacker*. Skema ini memiliki

keamanan dan *robustness* yang baik, tetapi kelemahannya adalah citra *host* mengalami modifikasi pada saat proses penyisipan.

Hou [6] dalam penelitiannya juga menggunakan *visual cryptography* untuk membagi citra *watermark* menjadi dua bagian. Bagian pertama disisipkan dalam beberapa piksel tertentu pada citra *host* dengan menurunkan nilai tingkat keabuannya. Selanjutnya *watermark* diekstraksi dengan menumpuk citra ber-*watermark* dan *share* kedua. Metode ini masih mengubah citra *host* selama proses *embedding* dan menunjukkan ketahanan yang lemah terhadap serangan geometris.

Di sisi lain untuk meningkatkan *robustness*, beberapa penelitian mengimplementasikan *watermarking* pada domain transformasi. *Watermarking* yang diimplementasikan pada *fractional Fourier* domain dapat meningkatkan ketahanan terhadap derau [7]. Lou [8] mengusulkan skema perlindungan hak cipta menggunakan *discrete wavelet transform* (DWT) dan *visual cryptography*. Fitur cira *host* diambil dengan kunci rahasia *K* dan hubungan antara koefisien wavelet *low* dan *middle sub-band*. Citra rahasia dihasilkan dengan bantuan *watermark* dan fitur citra *host*. Skema ini menunjukkan ketahanan yang baik terhadap berbagai serangan. Namun, hasil analisis yang dilakukan oleh Chen [9] terhadap skema tersebut menunjukkan bahwa skema ini tidak aman karena verifikasi *watermark* dapat dilakukan pada citra lain dengan kunci yang sama.

2.3. Citra Digital

Citra dapat didefinisikan sebagai fungsi intensitas cahaya dua dimensi $f(x, y)$, dimana x dan y menyatakan koordinat spasial dan nilai f pada sembarang titik (x, y) sebanding dengan skala kecerahan dari citra pada titik tersebut. Dalam bidang pengolahan citra, citra yang diolah adalah citra digital, yaitu citra kontinyu yang telah diubah ke dalam bentuk diskrit, baik koordiant ruang maupun intensitas cahayanya.

Citra digital $f(x, y)$ direpresentasikan sebagai sebuah matriks yang indeks baris dan kolomnya mengidentifikasikan

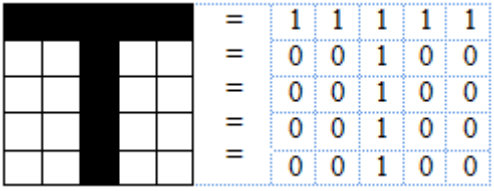
sebuah titik pada citra dan nilai dari elemen matriks yang bersangkutan merupakan tingkat warna pada titik tersebut. Elemen tersebut disebut dengan elemen citra atau piksel. Piksel dapat didefinisikan sebagai elemen terkecil dari sebuah citra digital yang menentukan resolusi citra tersebut. Semakin tinggi resolusi yang dihasilkan, semakin banyak jumlah pikselnya. Gambar 2.3 menunjukkan representasi citra digital yang berupa matriks dengan ukuran $M \times N$ [10]

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \cdots & & & \cdots \\ \cdots & & & \cdots \\ f(M-1,0) & f(M-1,1) & \ldots & f(M-1,N-1) \end{bmatrix}$$

Gambar 2.3 Representasi Citra Digital

2.3.1. Citra Biner

Citra biner, yaitu citra yang hanya terdiri atas dua warna, yaitu hitam dan putih. Oleh karena itu, setiap piksel pada citra biner cukup direpresentasikan dengan 1 bit. Citra biner hanya mempunyai dua nilai derajat keabuan yakni nol (0) untuk hitam dan satu (1) untuk putih [11]. Gambar 2.4 menunjukkan representasi citra biner.

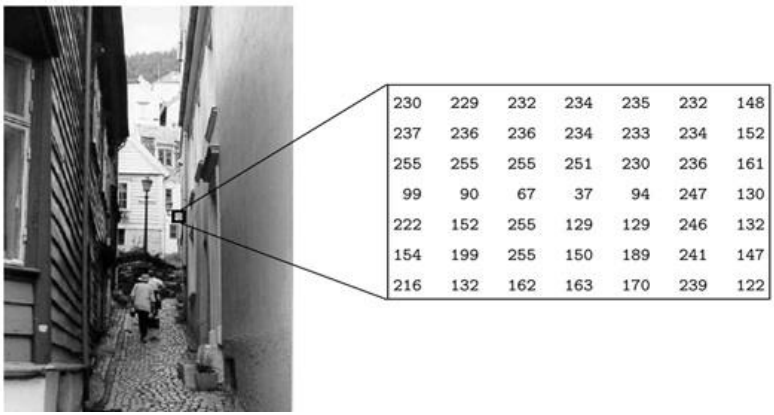


Gambar 2.4 Representasi Citra Biner

2.3.2. Citra *Grayscale*

Citra *grayscale* adalah citra digital yang hanya berisi informasi kecerahan tanpa memiliki unsur warna. Masing-masing piksel pada citra *grayscale* mengandung informasi banyaknya cahaya yang diterima pada titik itu. Nilai piksel pada citra *grayscale* biasanya memiliki kedalaman 8 bit, sehingga berada pada rentang 0-255. Semakin mendekati 0, maka warna semakin gelap yang menandakan semakin sedikit cahaya yang diterima. Sedangkan semakin mendekati 255, maka warna semakin cerah yang menandakan semakin banyak cahaya yang diterima.

Pada citra *grayscale* nilai setiap piksel komponen *red*=*green*=*blue*. Masing-masing warna dasar *red*, *green* dan *blue* hanya diambil nilai kecerahannya saja untuk membentuk nilai kecerahan pada citra *grayscale*. Pada warna hitam nilai *red*=*green*=*blue*=0, sedangkan pada warna putih nilai *red*=*green*=*blue*=255. Gambar 2.5 menunjukkan contoh representasi citra *grayscale*.



Gambar 2.5 Representasi Citra *Grayscale*

2.4. Transformasi Citra

Transformasi merupakan suatu proses yang digunakan untuk merepresentasikan suatu sinyal dari suatu domain ke domain lainnya. Pada proses *watermarking*, transformasi citra ke dalam domain frekuensi dapat dimanfaatkan untuk meningkatkan ketahanan *watermark*. Artinya *watermark* yang disisipkan ke dalam koefisien transformasi memiliki ketahanan yang lebih baik terhadap manipulasi citra.

Misalnya, algoritma *watermarking* yang menggunakan tranformasi DFT (*Discrete Fourier Transform*) tahan terhadap operasi pergeseran karena pergeseran dalam ranah spasial tidak mempunyai pengaruh terhadap magnitudo DFT. Ketahanan terhadap operasi geometri (seperti penskalaan, rotasi, atau pergeseran) dapat diperoleh dalam ranah *tranform* karena ranah *transform* dapat dirancang sedemikian sehingga *invariant* terhadap sekumpulan transformasi tertentu. Sedangkan ketahanan terhadap manipulasi *cropping* dapat diperoleh jika *watermark* disebar di antara seluruh komponen frekuensi.

2.4.1. *Fractional Fourier Transform* (FRFT)

Fractional Fourier transform (FRFT) merupakan suatu model transformasi yang memindahkan domain spasial/waktu menjadi domain frekuensi. FRFT merupakan generalisasi dari transformasi Fourier. Transformasi Fourier dari sebuah sinyal dapat diinterpretasikan sebagai sebuah rotasi sinyal terhadap sudut $\alpha = \pi/2$ dalam bidang frekuensi-waktu. Maka FRFT sebagai bentuk generalisasi dari transformasi Fourier dapat dikatakan sebagai sebuah operator linear yang berkorespondensi terhadap rotasi sinyal yang melewati sudut (α) yang tidak tetap [2].

FRFT banyak digunakan untuk transformasi karena memiliki fleksibilitas sudut rotasi yang dapat disesuaikan pada segala situasi. Pada tugas akhir ini FRFT digunakan untuk meningkatkan ketahanan *watermark* dari berbagai kemungkinan gangguan.

$$K_\alpha(t,u) = \begin{cases} \sqrt{\frac{1-j\cot\alpha}{2\pi}} \times e^{j\left(\frac{u^2+t^2}{2}\cot\alpha - tu\csc\alpha\right)} & if\ \alpha \neq n\,\pi \\ \delta(t-u) & if\ \alpha = n\,2\pi \\ \delta(t+u) & if\ \alpha + \pi \neq n\ \pi \end{cases}$$
$$= \sum_{n=0}^{\infty} e^{-j\alpha n} H_n(t) H_n(u) \qquad (2.1)$$

FRFT didefinisikan dengan bantuan transformasi kernel K_α seperti Persamaan 2.1. Di mana α sebagai sudut rotasi sinyal ter-transformasi. Dekomposisi *eigen* dari kernel FRFT juga dapat diinterpretasikan sebagai fungsi Hermite-Gaussian dengan $e^{-j\alpha n}$ sebagai *eigenvalue* dari FRFT kontinyu [3].

Fungsi FRFT untuk sinyal $x(t)$ dengan K_α sebagai transformasi kernel ditunjukkan pada Persamaan 2.2.

$$X_\alpha(u) = \int_{-\infty}^{+\infty} x(t) K_\alpha(t,u)\,dt \qquad (2.2)$$

Beberapa sifat-sifat yang dimiliki oleh *fractional Fourier transform* (FRFT) antar lain sebagai berikut:

1. Operator Identitas
   FRFT dengan orde $\alpha = 0$ atau $\alpha = 2\pi$ memberikan hasil yang sama dengan sinyal itu sendiri. Sifat ini ditunjukkan dengan Persamaan 2.3.
   $$F^0[f(x)] = F^{2\pi}[f(x)] = f(x) \qquad (2.3)$$
2. *Fourier Transform Operator* : $F^{\pi/2}$
   FRFT dengan orde $\alpha = \pi/2$ memberikan hasil yang sama dengan transformasi Fourier.
3. *Succesive application of FRFT*
   Succesive application dari FRFT ekuivalen terhadap transformasi tunggal di mana ordenya sama dengan jumlah dari masing-masing orde. Sifat ini ditunjukkan dengan Persamaan 2.4.
   $$F^\alpha\left(F^\beta[f(x)]\right) = F^{\alpha+\beta}[f(x)] \qquad (2.4)$$

4. Operasi Invers
   FRFT dengan orde $-\alpha$ memberikan hasil sebagai fungsi invers. Sifat ini ditunjukkan dengan Persamaan 2.5.
   $f(x) = F^{-\alpha}\left[F^{\alpha}[f(x)]\right]$ (2.5)

Dengan F merupakan operasi FRFT pada bidang frekuensi-waktu. Parameter α dan β adalah sudut rotasi antara sinyal ter-transformasi dan sumbu waktu pada bidang frekuensi-waktu.

FrFT dapat digunakan untuk mentransformasi data satu dimensi maupun dua dimensi. Citra merupakan data dua dimensi, sehingga untuk melakukan transformasi citra digunakan DFRFT 2 dimensi.

2.4.2. *Discrete Fractional Fourier Transform* (DFRFT)

Seperti FRFT, DFRFT merupakan bentuk generalisasi dari *Discrete Fourier Transform* (DFT). *Discrete Fourier transform* (DFT) adalah model transformasi Fourier yang digunakan pada fungsi diskrit, dan hasilnya juga diskrit. DFT memungkinkan digunakan untuk menganalisa, memanipulasi dan mensintesis sinyal yang tidak mungkin dapat dilakukan dalam pemrosesan sinyal analog.

DFT berasal dari transformasi Fourier kontinyu. DFT sebagai *discrete frequency domain sequence* $x(m)$ dinyatakan dengan Persamaan 2.6. Di mana $x(n)$ adalah sekuens diskrit nilai sampel dalam fungsi waktu dari variabel kontinyu $x(t)$.

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi nm}{N}} \quad (2.6)$$

DFRFT sebagai bentuk diskrit dari FRFT kontinyu harus tetap memiliki sifat rotasi yang sama dengan FRFT kontinyu. Sifat rotasi ini dapat diperoleh dengan memanfaatkan pangkat dari matriks kernel dalam kasus diskrit. Sehingga untuk menghitung DFRFT diperlukan sebagian kecil pangkat dari matriks kernel DFT [3].

Menurut Candan [12], matriks kernel untuk menghitung DFRFT dengan orde *a* dapat didefinisikan dengan Persamaan 2.7. Di mana U_k adalah himpunan *orthonormal eigenvector* tidak tetap dari matriks DFT yang berukuran $N \times N$ dan $\mu = \{0,1,2,\dots,N-2,(N-(N)_2)\}$. Orde a dapat diartikan sebagai pangkat dari *eigenvalue* $\lambda_{k=}e^{-j(\pi/2)k}$.

$$F^a[m,n] = \sum_{k\varepsilon\mu} U_k[m]\,(\lambda_k)^a U_k[n] = \sum_{k\varepsilon\mu} U_k[m] e^{-j(\pi/2)ka}\, U_k[n] \tag{2.7}$$

Karena matriks DFT hanya memiliki empat eigenvalue ($e^{-j(\pi/2)k} \in \{1,-1,-j,j\}$), maka *real eigenvector* (U) dari matriks kernel DFT didapatkan dengan menghitung *eigenvector* dari operasi komutatif matriks S dan F ($SF = FS$). F adalah matriks DFT. Sedangkan matriks S didefinisikan sebagai bagian dari fungsi Hermite-Gaussian diskrit.

Algoritma untuk mendapatkan *eigenvector* (U) melalui pendekatan matriks *S* dijabarkan sebagai berikut:

Masukan : panjang vektor f (N), orde matriks DFT (p)
Keluaran : himpunan *eigenvector* (U)

1. Membangkitkan matriks *S*
   Matriks S dengan orde p =2 didefinisikan dengan Persamaan 2.8. Di mana $\omega = 2\pi/N$ dan *N* merupakan ukuran matriks kernel DFT.

$$S = \begin{pmatrix} 2 & 1 & 0 & \dots & 0 & 1 \\ 1 & 2\cos\omega & 1 & \dots & 0 & 0 \\ 0 & 1 & 2\cos 2\omega & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0 \\ 1 & 0 & 0 & \cdots & 1 & 2\cos(N-1)\omega \end{pmatrix} \tag{2.8}$$

2. Membangkitkan matriks *P*
   Eigenvector yang terbentuk dari matriks *S* merupakan vektor *even* atau *odd*. Sehingga pencarian *eigenvector* dibatasi pada

komponen *even* dan *odd* dari matriks S. Oleh sebab itu digunakan matriks *P* yang mendekomposisi sebuah vektor $f[n]$ ke dalam komponen *odd* dan *even*. Komponen *even* dipetakan pada $\left\lfloor\left(\frac{N}{2}\right)+1\right\rfloor$ komponen pertama dan sisanya untuk komponen *odd*. Contoh matriks *P* untuk *N*=5 ditunjukkan oleh Persamaan 2.9.

$$P=\frac{1}{\sqrt{2}}\begin{pmatrix}\sqrt{2} & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 1\\ 0 & 0 & 1 & 1 & 0\\ 0 & 0 & 1 & -1 & 0\\ 0 & 1 & 0 & 0 & 1\end{pmatrix} \tag{2.9}$$

3. Dekomposisi dari matriks *Ev* dan *Od*
   Matriks *Ev* dan *Odd* dapat ditentukan dengan melakukan dekomposisi matriks melalui Persamaan 2.10.

$$PSP^{-1}=PSP=\begin{pmatrix}Ev & 0\\ 0 & Od\end{pmatrix} \tag{2.10}$$

4. Menghitung *eigenvector* dari matriks *Ev* dan *Od* dan mengurutkannya berdasarkan dari *eigenvalue* yang terbesar. Menotasikan *eigenvector* dari matriks *Ev* dan *Od* yang sudah urut sebagai e_k dan o_k.
5. Membuat matriks *U* berukuran $N \times N$ yang berisi e_k dan o_k dan vektor *zero* sesuai dengan Persamaan 2.11 dan 2.12.

$$U_{2k}[n]=P[{e_k}^T|0\ldots 0]^T \tag{2.11}$$
$$U_{2k+1}[n]=P[0\ldots 0|{o_k}^T]^T \tag{2.12}$$

Setelah transformasi kernel ditentukan, maka fungsi DFRFT untuk vektor $f[n]$ dengan orde a dapat didefinisikan menggunakan matriks kernel DFT F^a sesuai dengan Persamaan 2.13.

$$f_a=F^a f \tag{2.13}$$

2.4.3. *Two Dimensional Discrete Fractional Fourier Transform* (2D-DFRFT)

Sebuah titik (m, n) untuk transformasi diskirit 2 dimensi didefinisikan menggunakan kernel dengan Persamaan 2.14. Dengan $K(p, q, m, n)$ sebagai transformasi kernel 2D.

$$X(m,n) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} x(p,q) K(p,q,m,n) \tag{2.14}$$

Pada transformasi FRFT 2 dimensi, transformasi kernel dapat didefiniskan secara terpisah. Hal yang sama juga berlaku pada DFRFT 2 dimensi. Sehingga transformasi kernel dari 2D-DFRFT dapat dijabarkan seperti Persamaan 2.15.

$$K_{(\alpha,\beta)} = K_\alpha \otimes K_\beta \tag{2.15}$$

Di mana K_α dan K_β masing-masing merupakan transformasi kernel 1D-DFRFT. Kedua parameter DFRFT, α dan β menunjukkan orde masing-masing transformasi. Karena citra merupakan matriks 2D, maka transformasi DFRFT pada citra dapat diimplementasikan dengan komputasi pada vektor baris dan kolom. Sehingga fungsi 2D-DFRFT dapat dihitung dengan Persamaan 2.16.

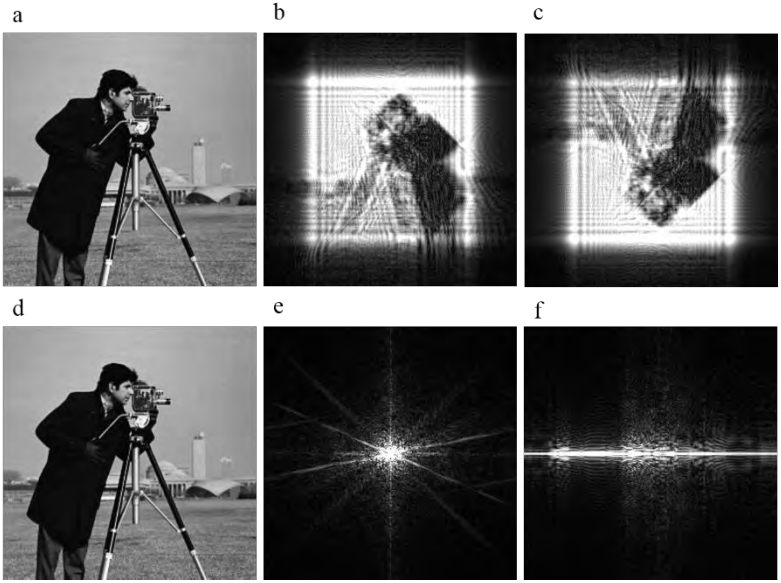
$$X(m,n) = \sum_{p=0}^{M-1} \left[\sum_{q=0}^{N-1} x(p,q) K_\beta(q,n) \right] K_\alpha(p,m) \tag{2.16}$$

Fungsi forward dan invers dari 2D-DFRFT dengan (α, β) sebagai parameter dan K_α, K_β sebagai kernel 1D-DFRFT dirumuskan dengan Persamaan 2.17 dan 2.18.

$$X_{(\alpha,\beta)}(m,n) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} x(p,q) K_{(\alpha,\beta)}(p,q,m,n) \tag{2.17}$$

$$x(p,q) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} X_{(\alpha,\beta)}(m,n) K_{(-\alpha,-\beta)}(p,q,m,n) \tag{2.18}$$

Implementasi 2D-DFRFT pada citra digital dengan berbagai parameter (α, β) memberikan hasil transformasi berbeda untuk tiap-tiap sudut dengan rentang 0 - π/2 [3]. Ini menunjukkan bahwa setiap parameter yang digunakan memiliki sensitifitas yang dapat digunakan sebagai kunci untuk melakukan transformasi.



Gambar 2.6 (a) Citra asli. (b) FRFT (3/2, 1/2). (c) FRFT (3/2, 5/2). (d) Invers FRFT (3/2, 1/2) dari Gambar (b). (e) Invers FRFT (-3/2, -1/2) dari Gambar (b) (f) Invers FRFT (3/2, -1/2) dari Gambar (b).

Pada Gambar 2.6 (b) dan (c) ditunjukkan hasil transformasi menggunakan 2D-DFRFT dengan orde transformasi yang berbeda. Dari gambar tersebut dapat dilihat perbedaan yang dihasilkan dari proses transformasi. Gambar 2.6 (d) menunjukkan hasil invers

transformasi menggunakan parameter yang benar. Sedangkan Gambar 2.6 (e) dan (f) menunjukkan hasil invers transformasi menggunakan parameter yang salah.

Transformasi 2D-DFRFT juga memiliki beberapa sifat khusus pada sudut-dudut tertentu. Di bawah ini merupakan beberapa sifat khusus dari transformasi 2D-DFRFT terhadap sudut-sudut tertentu, yaitu:

1. Jika $\alpha = \beta = \pi/2$ maka 2D-DFRFT memiliki kinerja yang sama dengan 2D-DFT.
2. Jika $\alpha = \pi/2$ dan $\beta = 0$ maka 2D-DFRFT hanya melakukan transformasi baris.
3. Jika $\alpha = 0$dan $\beta = \pi/2$ maka 2D-DFRFT hanya melakukan transformasi kolom.
4. Jika $\alpha = \beta = 0$ maka 2D-DFRFT melakukan *identity transform*.

2.5. *Singular Value Decomposition* (SVD)

Singular Value Decomposition (SVD) merupakan suatu teknik yang handal dalam melakukan berbagai analisis dan komputasi matriks, yaitu dengan membongkar struktur geometrinya, sehingga dapat diketahui beberapa sifat penting dari matriks tersebut. Sebuah matriks yang direpresentasikan dengan SVD akan didekomposisi menjadi 3 komponen matriks, yaitu matriks vektor singular kiri, matriks nilai singular, dan matriks vektor singular kanan. Jika diketahui suatu matriks adalah matriks A berukuran n×n dengan rank r > 0 , maka dekomposisi dari matriks A dinyatakan oleh [13] dengan Persamaan 2.19.

$$A = U * S * V^T \tag{2.19}$$

Rank (r) menyatakan banyaknya jumlah baris atau kolom yang saling independen antara baris atau kolom lainnya dalam suatu matriks. Komponen U dan V merupakan matriks *orthogonal* berukuran n×n, sedangkan S adalah matriks diagonal berukuran r×r

yang elemen diagonalnya merupakan akar positif dari *eigenvalue* matriks A.

Secara umum algoritma dekomposisi nilai singular adalah sebagai berikut:

Masukan : matrikss A

Keluaran : matrikss ortogonal U, V dan matrikss singular S sehingga $A = USV^T$.

1. Membentuk matrikss A^TA dengan nilai eigen λ_i untuk setiap $1 \leq i \leq n$, maka nilai singular (σ_i) matriks A dihitung dengan Persamaan 2.20.
   $$\sigma_i = \sqrt[2]{\lambda_i} \quad (2.20)$$
2. Membentuk matriks diagonal S sesuai dengan Persamaan 2.21.
   $$S = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n \end{bmatrix} \quad (2.21)$$
3. Mencari himpunan vektor eigen dari matriks A^TA. Misalkan $\{v_1, v_2, \dots, v_n\}$ merupakan vektor-vektor eigen matriks A^TA dengan v_1 sebagia vektor eigen yang bersesuaian dengan nilai λ_i.
4. Membentuk matriks ortogonal V sesuai dengan Persamaan 2.22.
   $$V = [v_1\ v_2\ \dots\ v_n] \quad (2.22)$$
5. Membentuk himpunan vektor $\{u_1, u_2, \dots, u_n\}$ untuk setiap $1 \leq i \leq n$ dengan Persamaan 2.23.
   $$u_i = \frac{1}{\sigma_i} A v_i \quad (2.23)$$
6. Membentuk matriks ortogonal sesuai dengan Persamaan 2.24.
   $$U = [u_1\ u_2\ \dots\ u_n] \quad (2.24)$$
7. Membentuk dekomposisi SVD sesuai Persamaan 2.19, yaitu $A = USV^T$.

Proses di atas disebut sebagai dekomposisi nilai singular. Nilai dari S disebut sebagai nilai-nilai singular dari A, kolom-kolom dari U merupakan vektor-vektor singular kiri dari A dan kolom-kolom dari V disebut sebagai vektor-vektor singular kanan dari A.

Ketika *A* direpresentasikan dengan SVD, maka akan terbentuk sejumlah *basis image* yang menyusunnya sebanyak jumlah nilai singular yang terbentuk. *Basis image* pertama (yang berasal dari nilai singular terbesar) merepresentasikan komponen-komponen berfrekuensi rendah dari citra *A*. *Basis image* kedua merepresentasikan komponen-komponen dengan frekuensi yang lebih tinggi dan seterusnya. Dengan kata lain, semakin tinggi orde *basis image*-nya semakin tinggi pula frekuensi komponen-komponen citra yang terkandung di dalamnya.

Nilai-nilai singular pada citra digital hanya terpengaruh sedikit meskipun citra tersebut sudah mengalami gangguan. Karena pada hakikatnya nilai-nilai singular mengandung sifat-sifat aljabar citra. Sehingga SVD dapat dimanfaatkan untuk meningkatkan kinerja algoritma *watermarking* citra digital. Penyisipan *watermark* dengan cara memodifikasi nilai singualar terbesar dari matriks *S* mampu menghasilkan citra yang tahan dari serangan geometri [13].

2.6. Visual Secret Sharing (VSS)

Visual secret sharing (VSS) [14] adalah konsep pembagian citra rahasia yang pertama kali dikenalkan oleh Naor dan Shamir. Skema ini dapat digunakan untuk menyembunyikan *watermark* dan mengklaim kepemilikan citra tanpa mengubah citra *host.* Dengan metode ini citra *watermark* tidak secara langsung disisipkan ke dalam citra *host*, tetapi penyisipan dilakukan dengan menggunakan pola acak dari citra biner yang disebut *share*. Beberapa penelitian telah membuktikan bahwa penerapan konsep VSS pada *watermarking* citra dapat memecahkan permasalahan yang sering muncul pada algoritma *watermarking* konvensional. Jika dibandingkan dengan teknik *watermarking* yang lain, metode ini memiliki tiga manfaat utama, yaitu efisiensi waktu, memberikan tingkat keamanan yang tinggi dan mengurangi *trade off* diantara *imperceptibility*, *capacity* dan *robustness* [4]. Kelebihan pendekatan teknik *watermarking* ini juga dapat

digunakan untuk melakukan perlindungan citra sensitif seperti citra medis, citra militer dan citra satelit.

Skema ini merupakan jenis baru skema *visual cryptography* yang dapat membaca sandi gambar tersembunyi tanpa perhitungan kriptografi. Skema ini memiliki tingkat keamanan yang baik dan cukup mudah untuk diimplementasikan. Skema ini membagi citra menjadi n *share* berbeda. Setiap citra dapat diperoleh kembali dengan menumpuk $k(k \leq n)$ atau lebih dari k *share*, sehingga sangat tepat untuk algoritma *blind watermarking*.

Algoritma penyisipan *watermark* yang berbasis VSS dilakukan mengikuti alur kerja berikut:

Masukan : citra *grayscale* sebagai citra *host* dan citra biner sebagai citra *watermark*.

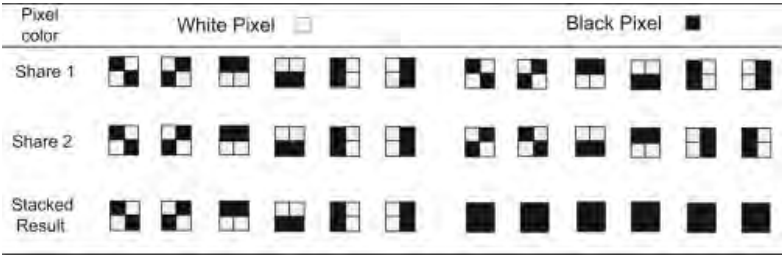
Keluaran : citra biner acak (*private share*).

1. Membentuk vektor fitur dari citra *host.*
2. Mengubah vektor fitur menjadi matriks biner.
3. Membentuk *public share* yang berupa citra biner acak dari matriks biner dengan sebuah kunci rahasia dan metode tambahan.
4. Menyisipkan citra *watermark* ke dalam *public share* manggunakan *codebook* VSS, sehingga terbentuk citra biner acak yang disebut *private share*.

Private share selanjutnya didaftarkan dengan *Certified Authority* (CA) untuk keperluan lebih lanjut. Selama verifikasi *copyright*, proses yang sama digunakan untuk mengekstrak *public share* dari citra yang diklaim menggunakan kunci rahasia yang sama. Selanjutnya *public share* dikombinasikan dengan *private share* yang sudah disimpan untuk mengekstraksi *watermark* yang telah disisipkan. Keseluruhan proses *watermarking* ini dilakukan tanpa mengubah citra *host* dan oleh sebab itu kualitas citra *host* tetap tinggi [4].

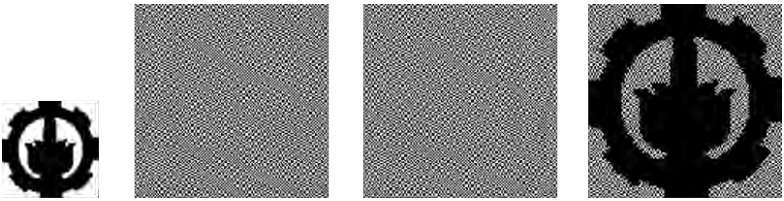
Dalam tugas akhir ini skema VSS yang digunakan adalah skema VSS (2, 2). Skema VSS (n, n) adalah skema yang paling aman dan paling mudah dalam manajemen kunci. Pada proses

enkripsi sebuah citra *watermark* dengan ukuran $m \times n$ dapat dibagi menjadi dua *share*, yaitu *master share* dan *ownership share*. *Master share* dibangun dari fitur citra *host*, sedangkan *ownership share* dibangun dari *master share* dan *citra watermark* menggunakan teknik *visual cryptography* sesuai skema VSS $(2, 2)$.



Gambar 2.7 Skema VSS (2, 2)

Gambar 2.7 menunjukkan skema VSS $(2, 2)$ yang digunakan pada tugas akhir ini. Setiap *share* dibentuk dengan mengganti setiap piksel citra *watermark* menjadi blok piksel 2×2, sehingga masang-masing *share* berukuran $2m \times 2n$. Setiap piksel putih dari citra *watermark* direpresentasikan ke dalam blok piksel yang identik. Sedangkan setiap piksel hitam direpresentasikan ke dalam blok piksel yang saling berkomplemen.



(a) (b) (c) (d)

Gambar 2.8 Contoh VSS (2, 2) (a) Citra *Watermark* (b) *Master Share* (c) *Ownership Share* (d) Hasil Ekstraksi (Citra *Stack*)

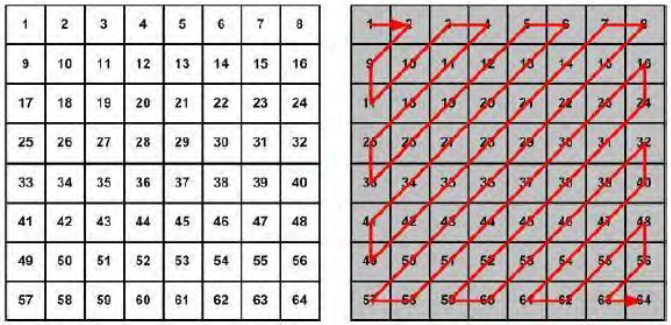
Proses ekstraksi citra *watermark* pada skema ini dilakukan dengan cara menumpuk *master share* dari citra yang diklaim dan *ownership share* yang sudah disimpan. Hasil algoritma *blind watermarking* menggunakan konsep VSS $(2, 2)$ ditunjukkan oleh Gambar 2.8. Citra hasil ekstraksi seperti pada Gambar 2.8 (d) berukuran $2m \times 2n$, sehingga citra harus dikecilkan kembali seperti ukuran citra *watermark* sebelumnya.

2.7. Algoritma Pengacakan

Penanganan sistem keamanan pada algoritma *blind watermarking* yang diusulkan salah satunya adalah dengan melakukan pengacakan posisi blok-blok piksel. Algoritma yang digunakan untuk melakukan pengacakan blok-blok piksel yaitu zigzag *scanning* dan *cat map.*

2.7.1. Zigzag Scanning

Zigzag *scanning* merupakan suatu cara untuk membaca matriks dengan cara zigzag. Matriks yang dibaca biasanya berbentuk persegi $(n \times n)$. Sebuah matriks dibaca secara zigzag dari kiri atas sampai kanan bawah seperti yang ditunjukkan pada Gambar 2.9.



Gambar 2.9 Representasi Zigzag *Scanning*

Zigzag *scanning* dilakukan pada kumpulan blok-blok piksel yang berukuran $(n \times n)$. Masing-masing mewakili posisi sebuah blok piksel. Hasil zigzag *scanning* berupa *sequence* blok-blok piksel. Selanjutnya *sequence* blok-blok piksel ini di *reshape* menjadi ukuran $(n \times n)$ yang menghasilkan blok-blok piksel dengan posisi berbeda dari sebelumnya.

2.7.2. Cat Map

Cat Map adalah fungsi *chaos* dwimatra yang mentransformasikan koordinat (x, y) dari citra berukuran $N \times N$ ke koordinat baru (x', y') [15]. Persamaan *cat map* untuk mengacak blok-blok piksel ditunjukkan oleh persamaan 2.25. Persamaan tersebut merupakan persamaan Arnold *cat map* yang telah dikembangkan.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & a \\ b & ab+1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} mod\ N \tag{2.25}$$

Cat map bersifat *reversible*, yaitu citra hasil transformasinya dapat dikembalikan ke citra semula dengan. Persamaan *cat map* untuk mengembalikan hasil transformasi ke citra semula ditunjukkan oleh Persamaan 2.26.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & a \\ b & ab+1 \end{bmatrix}^{-1} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} mod\ N \tag{2.26}$$

Jumlah blok yang digunakan untuk masukan harus berukuran $N \times N$. (x_i, y_i) Merupakan koordinat awal blok piksel sedangkan (x_{i+1}, y_{i+1}) merupakan koordinat baru hasil *cat map*. Sedangkan parameter a dan b adalah integer positif sembarang yang dapat dianggap sebagai kunci rahasia.

Hasil dari *cat map* adalah pergeseran ke dalam arah sumbu y, kemudian ke dalam arah sumbu x. Semua hasilnya di modulo N agar hasilnya tetap pada area citra. Dengan melakukan iterasi berkali-kali maka hasilnya semakin acak. Tetapi setelah iterasi tertentu hasilnya kembali pada koordinat semula. Gambar 2.10

menunjukkan hasil pengacakan blok-blok piksel citra menggunakan *cat map* pada iterasi pertama.

Gambar 2.10 Citra Asli dan Citra Hasil Pengacakan dengan *Cat Map* Iterasi Pertama

Pada tugas akhir ini *cat map* digunakan untuk mengacak posisi blok-blok piksel pada citra setelah dilakukan zigzag *scanning*. Pengacakan dilakukan untuk meningkatkan sistem keamanan dari algoritma *blind watermarking* yang diusulkan.

2.8. Penilaian Kualitas Citra

Penilaian kualitas citra pada uji coba tugas akhir ini dilakukan secara obyektif dengan menggunakan besaran PSNR dan NC.

2.8.1. Peak Signal to Noise Ratio (PSNR)

PSNR (*Peak Signal to Noise Ratio*) merupakan nilai perbandingan antara harga maksimum warna pada citra hasil proses dengan kuantitas gangguan (derau) yang dinyatakan dalam satuan desibel (dB). Derau yang dimaksud adalah akar rata-rata kuadrat nilai kesalahan $\sqrt{MSE}$. Pada tugas akhir ini, PSNR digunakan untuk mengetahui kualitas citra yang diganggu dibandingkan dengan citra yang asli. Semakin tinggi nilai PSNR

yang dihasilkan semakin baik. Secara matematis, PSNR dapat dirumuskan dalam Persamaan 2.27.

$$PSNR = 10 \log_{10}\left(\frac{255^2}{MSE}\right) \quad (2.27)$$

$$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left(H_{i,j} - \widehat{H}_{i,j}\right)^2 \quad (2.28)$$

Nilai 255 berarti sebagai batas atas nilai piksel pada citra 8 bit (0-255). MSE (*Mean Square Error*) adalah rata-rata kuadrat nilai kesalahan antara citra asli (citra *host*) dengan citra yang sudah mengalami gangguan yang secara matematis dapat dirumuskan dalam Persamaan 2.28. Di mana *M* dan *N* merupakan lebar dan tinggi citra, $H_{i,j}$ dan $\widehat{H}_{i,j}$ merupakan nilai dari piksel pada koordinat (i,j) pada citra *host* dan citra yang sudah diganggu.

2.8.2. Normalized Correlation (NC)

Normalized correlation (NC) merupakan nilai yang menunjukkan tingkat kemiripan antara dua buah citra yang dinyatakan dalam rentang 0-1. Pada tugas akhir ini NC digunakan untuk menghitung tingkat kemiripan antara *watermark* hasil ekstraksi dengan citra *watermark* asli. Secara matematis, NC dapat dirumuskan dalam Persamaan 2.29.

$$NC = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\overline{w_{i,j} \oplus w'_{i,j}}}{m \times n} \quad (2.29)$$

$w_{i,j}$ merupakan citra *watermark* asli sedangkan $w'_{i,j}$ adalah citra *watermark* hasil ekstraksi, $\oplus$ dinotasikan sebagai operasi exclusive-or (XOR) dan $m \times n$ merupakan ukuran citra.

Halaman ini sengaja dikosongkan

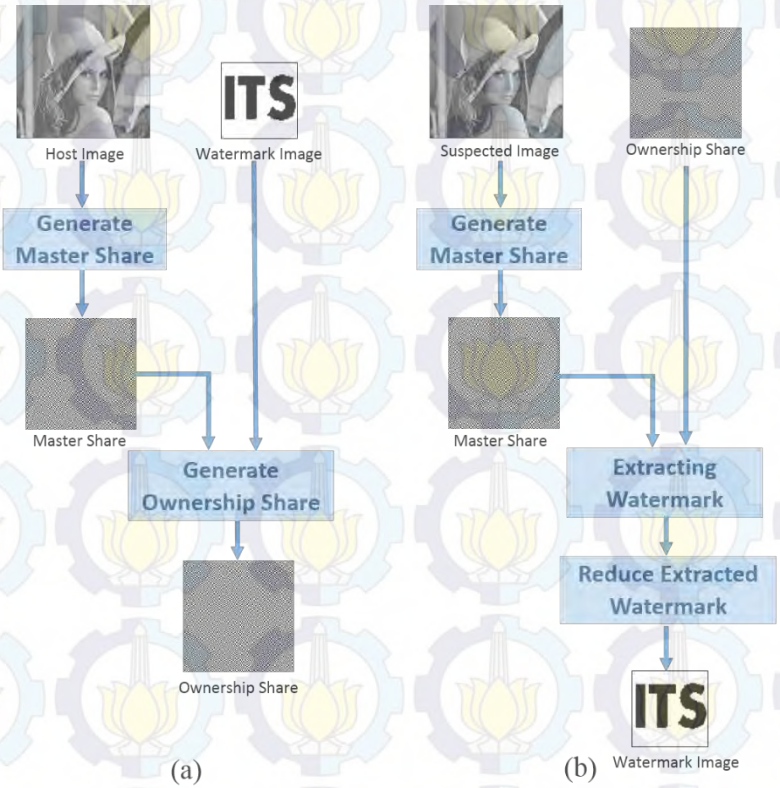
BAB III
PERANCANGAN PERANGKAT LUNAK

Bab ini membahas secara terperinci dan menyeluruh mengenai perancangan perangkat lunak *watermarking* citra. Perancangan dibagi menjadi dua proses utama, yaitu proses registrasi kedua *share* dan proses ekstraksi *watermark.* Masing-masing proses utama dibagi menjadi beberapa proses-proses kecil yang terlibat di dalamnya. Di awal bab akan dijelaskan desain metode secara umum dalam bentuk diagram alir, selanjutnya penjelasan lebih detail disajikan dalam bentuk *pseudocode*.

3.1. Desain Metode Secara Umum

Pada tugas akhir ini dibangun sebuah perangkat lunak *blind watermarking* yang dapat melakukan *watermarking* terhadap citra tanpa harus mengubah citra tersebut dan proses ekstraksi *watermark* dapat dilakukan dengan cara yang lebih sederhana. Sistem perangkat lunak *blind watermarking* ini dirancang menggunakan metode *Fractional Fourier Transform* dan teknik *visual cryptography*. Penggabungan kedua metode ini bertujuan agar citra *watermark* yang disisipkan memiliki sifat *robust* dan *secure* yang baik.

Sistem perangkat lunak *blind watermarking* ini terbagi menjadi dua proses utama, yaitu proses penyisipan (*embedding process*) dan proses ekstraksi (*extracting process*). Proses penyisipan membutuhkan dua data masukan, yaitu data masukan citra *host* berupa citra *grayscale* dan data masukan citra *watermark* berupa citra biner. Data keluaran yang dihasilkan berupa citra kunci (*ownership share*). Data keluaran tersebut selanjutnya digunakan sebagai data masukan pada proses ekstraksi. Hasil dari proses ekstraksi adalah citra *watermark* hasil proses ekstraksi yang mana kualitasnya tidak boleh berbeda jauh dengan citra *watermark* asli. Gambaran umum keseluruhan sistem ini ditunjukkan oleh Gambar 3.1.



Gambar 3.1 Gambaran Umum Keseluruhan Sistem (a) Proses Penyisipan (b) Proses Ekstraksi

Berbeda dengan algoritma *watermarking* tradisional yang menyisipkan *watermark* ke dalam citra, pada tugas akhir ini *watermark* disisipkan menggunakan skema *visual secret sharing* (VSS (2, 2)) yang membagi citra *watermark* (m×n) ke dalam 2 *share*. Kedua *share* tersebut adalah *master share* dan *ownership share*. Masing-masing *share* tersebut berukuran 2m×2n. *Master share* dibentuk dari matriks biner hasil pembentukan vektor fitur

citra *host*. Sedangkan *ownership share* dibentuk dari *master share* dan citra *watermark*.

Proses penyisipan dan ekstraksi *watermark* pada tugas akhir ini dilakukan dengan bantuan kedua *share* tersebut. Sehingga pada masing-masing proses utama terdapat subproses yang sama, yaitu pembentukan *master share*. Pembentukan *master share* secara lebih detil digambarkan oleh diagram alir pada Gambar 3.2.

Gambar 3.2 Diagram Alir Proses Pembentukan *Master Share*

Pada proses ekstraksi, citra *watermark* didapatkan kembali dengan cara menumpuk *master share* dan *ownership share*. Sehingga diperoleh hasil ekstraksi *watermark* yang berukuran 2m×2n atau dua kali lebih besar dari citra *watermark* yang disisipkan. Oleh sebab itu pada proses ekstraksi *watermark* terdapat tahapan untuk mengecilkan ukuran citra *watermark*.

3.2. Perancangan Data

Perancangan data merupakan bagian yang penting dalam pengoperasian perangkat lunak, karena tanpa data yang benar perangkat lunak tidak dapat beroperasi dengan benar. Data yang diperlukan dalam pengoperasian perangkat lunak adalah data masukan (*input*), data proses yang dibutuhkan dan dihasilkan selama proses eksekusi perangkat lunak, dan data keluaran (*output*) yang memberikan hasil proses pengoperasian perangkat lunak.

3.2.1. Data Masukan

Data masukan (*input*) merupakan data yang dimasukkan oleh pengguna pada sistem *blind watermarking* citra menggunakan FRFT dan teknik *visual cryptography*. Pada proses penyisipan *watermark* terdapat dua data masukan, yaitu citra *host* (H) dan citra *watermark* (w).

Gambar 3.3 Contoh Data Masukan Citra *Host*

Data masukan citra *host* berupa citra *grayscale* berukuran 512×512 yang diunduh secara acak dari *database* USC-SIPI. Adapun contoh citra yang digunakan sebagai data masukan citra *host* dapat dilihat pada Gambar 3.3.

Sedangkan data masukan citra *watermark* berupa citra biner (*monochrome*) berukuran 64×64. Adapun contoh citra yang digunakan sebagai data masukan citra *watermark* dapat dilihat pada Gambar 3.4.

Gambar 3.4 Contoh Data Masukan Citra *Watermark*

Pada proses ekstraksi juga terdapat dua data masukan. Data masukan pertama yaitu citra *host* yang diklaim (H') yang sudah mengalami gangguan seperti penambahan derau, kompresi JPEG, rotasi, *blurring*, *sharpening*, *resizing* dan *cropping*. Sedangkan data masukan kedua berupa citra *ownership share*, yaitu citra keluaran yang dihasilkan dari proses penyisipan *watermark* yang telah disimpan sebelumnya.

3.2.2. Data Proses

Pada subab ini dijelaskan bahwa data proses yang digunakan dalam perancangan perangkat lunak meliputi nama data (variabel dan fungsi), tipe data, dan penjelasannya. Daftar variabel dan fungsi yang digunakan untuk proses penyisipan ditunjukkan pada Tabel 3.1 sampai Tabel 3.6. Sedangkan daftar variabel dan fungsi yang digunakan untuk proses ekstraksi ditunjukkan pada Tabel 3.7 sampai Tabel 3.9.

Tabel 3.1 Daftar Variabel yang Digunakan pada Proses Penyisipan *Watermark* (Bagian 1)

No	Nama Variabel	Tipe Data	Penjelasan
1.	*A*	*double*	Kunci untuk parameter *Cat map*
2.	*a*	*double*	Orde transformasi untuk 1D-DFRFT
3.	*a_key*	*string*	Kunci untuk parameter *Cat map*
4.	*alpha*	*double*	Kunci untuk parameter 2D-DFRFT
5.	*alpha_key*	*string*	Kunci untuk parameter 2D-DFRFT
6.	*angles*	*double*	Matriks 1×2 yang berisi parameter 2D-DFRFT (α, β)
7.	*ansAkey*	*cell*	Hasil split variable *a_key*
8.	*ansAlpha*	*cell*	Hasil split variable *alpha*
9	*ansBeta*	*cell*	Hasil split variable *beta*
10.	*ansBkey*	*cell*	Hasil split variable *b_key*
11.	*avg*		Nilai rata-rata dari matriks singular (*m_singular*)
12.	*B*	*double*	Kunci untuk parameter *Cat map*
13.	*b_key*	*string*	Kunci untuk parameter *Cat map*
14.	*beta*	*double*	Kunci untuk parameter 2D-DFRFT
15	*beta_key*	*string*	Kunci untuk parameter 2D-DFRFT
16.	*block*	*cell*	*Cell* berukuran 128x128 dimana masing-masing *cell* berisi 8×8 piksel dari citra *host*

Tabel 3.2 Daftar Variabel yang Digunakan pada Proses Penyisipan *Watermark* (Bagian 2)

No	Nama Variabel	Tipe Data	Penjelasan
17.	*blockcatmap*	*cell*	*Cell* berukuran 128x128 dimana masing-masing cell berisi 8×8 piksel hasil operasi *Cat map*
18.	*blockfrft*	*double*	*Cell* berukuran 128x128 dimana masing-masing cell berisi 8×8 piksel hasil transformasi 2D-DFRFT
19.	*blockzigzag*	*cell*	*Cell* berukuran 128x128 dimana masing-masing cell berisi 8×8 piksel hasil operasi zigzag *scanning*
20.	*col_block*	*double*	Ukuran kolom untuk setiap blok-blok piksel
21.	*col_host*	*double*	Jumlah kolom piksel citra *host*
22..	*col_share*	*integer*	Jumlah kolom piksel citra *master share* dan *ownership share*
23.	*cur_col*	*double*	Indeks kolom yang sedang dioperasikan pada zigzag *scanning*
24.	*cur_index*	*double*	Indeks untuk menandai *sequence* hasil zigzag *scanning* yang sedang dioperasikan.
25.	*cur_row*	*double*	Indeks baris yang sedang dioperasikan pada zigzag *scanning*

Tabel 3.3 Daftar Variabel yang Digunakan pada Proses Penyisipan *Watermark* (Bagian 3)

No	Nama Variabel	Tipe Data	Penjelasan
26.	*E*	*double*	Matriks (N×1) yang berisi himpunan *eigenvector* untuk membentuk transformasi kernel 1D-DFRFT
27.	*E_saved*	*double*	*Eigenvector* yang sudah disimpan
28.	*ee*	*double*	*Eigenvalue* dari matriks *Ev*
29.	*eo*	*double*	*Eigenvalue* dari matriks *Odd*
30.	*even*		Variabel untuk menandai jenis data (ganjil atau genap)
31.	*f*	*double*	Vektor yang sedang dioperasikan 1D-DFRFT
32.	*host*	*uint8*	Citra *host* asli berupa citra *grayscale* berukuran 512×512
33.	*hslzigzag*	*cell*	Cell berukuran 1×4096 yang digunakan untuk menyimpan hasil pembacaan zigzag *scanning*
34.	*ind*	*double*	Indeks yang digunakan untuk mengubah posisi *eigenvector*
35.	*index*	*double*	Matriks 2×1 yang berisi koordinat baru hasil *cat map*
36.	*m*	*integer*	Jumlah baris piksel citra *watermark*
37.	*m_biner*	*double*	Matriks hasil binerisasi dari m_singular yang berukuran 64×64
38.	*m_master*	*double*	Matriks *master share* berukuran 128×128 yang dibangkitkan dari citra *host*

Tabel 3.4 Daftar Variabel yang Digunakan pada Proses Penyisipan *Watermark* (Bagian 4)

No	Nama Variabel	Tipe Data	Penjelasan
39.	*m_owner*	*double*	Matriks *ownership share* berukuran 128×128 yang dibangkitkan dari citra *host*
40.	*m_singular*	*double*	Matriks berukuran 64×64 yang berisi nilai singular pertama dari tiap-tiap *blockfrft*
41.	*matrix*	*double*	Matriks yang akan dilakukan operasi 2D-DFRFT
42.	*N*	*double*	Jumlah kolom piksel citra *watermark*
43.	*N*	*double*	Panjang vektor *f*
44.	*p*	*double*	Penaksiran orde untuk pembentukan *eigenvector* (E)
45.	*P*	*double*	Matriks P yang digunakan untuk mendekomposisi komponen *even* dan *odd*
46.	*p_saved*	*double*	Penaksiran orde untuk pembentukan *eigenvector* (*E*) yang sudah disimpan
47.	*P1*		Sebagian dari matriks P yang bernilai negatif
48.	*r*	*double*	Nilai tengah dari panjang vektor
49.	*row_block*	*double*	Ukuran baris untuk setiap blok-blok piksel
50.	*row_host*	*double*	Jumlah baris piksel citra *host*
51.	*row_share*	*double*	Jumlah baris piksel citra *master share* dan *ownership share*

Tabel 3.5 Daftar Variabel yang Digunakan pada Proses Penyisipan *Watermark* (Bagian 5)

No	Nama Variabel	Tipe Data	Penjelasan
52.	*S*	*double*	Matriks S yang digunakan untuk membentuk *eigenvector* dari matriks DFT
53..	*secret*	*logical*	Citra *watermark* berukuran 64x64
54.	*shft*	*double*	Vektor *f* yang sudah dibalik posisinya
55.	*size_N*	*double*	Ukuran matriks persegi yang akan dilakukan *Cat map*
56.	*tm*	*second*	Waktu mulai *running* program pembentukan *master share*
57.	*tMaster*	*second*	Total waktu *running* program pembentukan *master share*
58.	*to*	*second*	Waktu mulai *running* program pembentukan *ownership share*
59.	*tOwner*	*second*	Total waktu *running* program pembentukan *ownership share*
60.	*type*	*double*	Kunci untuk parameter VSS
61.	*ve*	*double*	*Eigenvector* dari matriks *Ev*
62.	*vo*	*double*	*Eigenvector* dari matriks *Odd*
63	*y*	*double*	Vektor yang hasil operasi 1D-DFRFT

Tabel 3.6 Daftar Fungsi yang Digunakan pada Proses Penyisipan *Watermark*

No	Nama Fungsi	Penjelasan
1.	*binaryMap*	Fungsi untuk melakukan pembentukan matriks biner
2.	*catmap*	Fungsi untuk melakukan *cat map*
3.	*dfrft1d*	Fungsi untuk melakukan transformasi 1D-DFRFT
4.	*frft2d*	Fungsi untuk melakukan transformasi 2D-DFRFT
5.	*mean2*	Fungsi untuk menghitung nilai rata-rata sebuah matriks
6.	*ones*	Fungsi untuk membentuk matriks yang benilai '1'
7.	*reshape*	Fungsi untuk membuat matriks (N×N) dari sebuah *sequence*
8.	*svds*	Fungsi untuk mendapatkan nilai singular matriks
9.	*tic*	Fungsi untuk mendapatkan waktu mulai *running* program
10.	*toc*	Fungsi untuk mendapatkan total waktu *running* program
11.	*vssmaster*	Fungsi untuk membangun citra *master share*
12.	*vssowner*	Fungsi untuk menyisipkan citra *watermark* (membangun citra *ownership share*)
13.	*zeros*	Fungsi untuk membentuk matriks yang benilai '0'
14.	*zigzag*	Fungsi untuk melakukan zigzag *scanning*

Tabel 3.7 Daftar Variabel yang Digunakan pada Proses Ekstraksi *Watermark* (Bagian 1)

No	Nama Variabel	Tipe Data	Penjelasan
1.	*blockSS*	*cell*	Cell berukuran 64×64 yang berisi blok-blok piksel dari *stcakIMG*
2.	*col_host*	*integer*	Jumlah baris piksel citra *host*
3.	*col_s*	*double*	Jumlah kolom piksel citra *stackIMG*
4.	*col_share*	*integer*	Jumlah kolom piksel citra *master share, ownership share dan stackIMG*
5.	*host_aksen*	*uint8*	Citra *host* yang sudah dimanipulasi berukuran 512×512
6.	*m*	*integer*	Jumlah baris piksel citra *watermark*
7.	*m_master*	*double*	Matriks *master share* berukuran 128×128 yang dibangkitkan dari citra *host* yang sudah dimanipulasi
8.	*m_owner*	*double*	Matriks *ownership share* berukuran 128×128
9.	*n*	*integer*	Jumlah kolom piksel citra *watermark*
10.	*nc_value*		Nilai hasil uji NC
11.	*reducedIMG*		Citra *watermark* hasil ekstraksi yang sudah dikecilkan
12.	*row_host*	*integer*	Jumlah kolom piksel citra *host*

Tabel 3.8 Daftar Variabel yang Digunakan pada Proses Ekstraksi *Watermark* (Bagian 2)

No	Nama Variabel	Tipe Data	Penjelasan
13.	*row_s*	*double*	Jumlah baris piksel citra *stackIMG*
14.	*row_share*	*integer*	Jumlah baris piksel citra *master share, ownership share dan stackIMG*
15.	*secret*	*logical*	Citra watermark berukuran 64x64
16.	*secret1*	*double*	Citra *watermark* asli
17.	*secret2*	*double*	Citra *watermark* hasil ekstraksi
18.	*size_secret*	*double*	Luas citra *watermark*
19.	*stackIMG*	*Uint8*	Citra *watermark* hasil proses *stacked* yang berukuran 128×128
20.	*sum*	*double*	Jumlah nilai dalam sebuah *blockSS*
21.	*te*	*second*	Waktu mulai *running* program ekstraksi citra *watermark*
22.	*tEkstrak*	*second*	Total waktu *running* program ekstraksi citra *watermark*
23.	*tm*	*second*	Waktu mulai *running* program pembentukan *master share*
24.	*tMaster*	*second*	Total waktu *running* program pembentukan *master share*

Tabel 3.9 Daftar Fungsi yang Digunakan Pada Proses Ekstraksi *Watermark*

No	Nama Fungsi	Penjelasan
1.	*fsum*	Fungsi untuk melakukan penjumlahan elemen-elemen matriks pada *blockSS*
2.	*nc*	Fungsi untuk melakukan uji NC
3.	*reduced*	Fungsi untuk mengecilkan ukuran citra *watermark* hasil ekstraksi
4.	*stacked*	Fungsi untuk mengekstraksi citra *watermark*

3.2.3. Data Keluaran

Data keluaran untuk proses penyisipan adalah citra kunci (*ownership share*) berupa citra biner berukuran $2m \times 2n$ yang mana $m \times n$ merupakan ukuran citra *watermark*. Citra *ownership share* ini selanjutnya harus diautentikasi dengan CA untuk keamanan lebih lanjut. Contoh hasil keluaran dari proses penyisipan ditunjukkan pada Gambar 3.5.

Gambar 3.5 Contoh Citra *Ownership share*

Sedangkan data keluaran untuk proses ekstraksi adalah citra *watermark* hasil ekstraski. Pada keadaan normal, citra *watermark* hasil ekstraski harus memiliki kualitas yang sama (tidak berbeda signifikan) dengan citra *watermark* yang disisipkan pada

proses sebelumnya. Adapun contoh citra yang merupakan citra *watermark* hasil ekstraksi dapat dilihat pada Gambar 3.6.

Gambar 3.6 Contoh Citra *Watermark* Hasil Ekstraksi

3.3. Perancangan Proses Penyisipan *Watermark*

Salah satu dari dua proses utama yang ada pada aplikasi ini adalah proses penyisipan citra *watermark*. Gambaran umum proses penyisipan citra *watermark* digambarkan pada subbab sebelumnya yaitu pada Gambar 3.1 (a). Dapat dilihat masukan pada proses penyisipan ada dua, yaitu citra *grayscale* sebagai citra *host* dan citra biner sebagai citra *watermark*. Sedangkan tahap-tahap yang dilakukan pada proses penyisipan *watermark* ditunjukkan pada Gambar 3.7.

Tahap pertama proses penyisipan *watermark* dimulai dengan pembentukan vektor fitur dari citra *host*. Vektor fitur adalah sebuah matriks biner yang merepresentasikan karakteristik citra *host* dan memiliki ukuran yang sama dengan citra *watermark*. Pembentukan vektor fitur dimulai dengan membagi citra *host* menjadi blok-blok sejumlah ukuran citra *watermark*. Jadi, jika ukuran citra *watermark* 64×64, maka citra *host* dibagi menjadi blok-blok *non-overlapping* sejumlah 64×64.

Selanjutnya blok-blok tersebut diacak posisinya menggunakan zigzag *scanning* dan *cat map*. Pengacakan dilakukan untuk menyebar posisi agar menghasilkan *ownership share* yang lebih tahan terhadap serangan. Masing-masing blok ditransformasi dengan FRFT dan didekomposisi menggunakan SVD. Sehingga terbentuk sebuah matriks berukuran 64×64 yang berisi nilai singular pertama dari masing-masing blok. Masing-masing nilai

singular yang tersimpan merepresentasikan karakteristik dari blok-blok piksel. Selanjutnya matriks nilai singular diubah menjadi matriks biner agar dapat digunakan pada proses pembentukan *master share*. Rangkaian proses pembentukan vektor fitur yang dilakukan secara bertahap bertujuan agar fitur yang didapatkan benar-benar merepresentasikan karakter citra secara keseluruhan.

Gambar 3.7 Diagram Alir Proses Penyisipan *Watermark*

Tahap selanjutnya adalah pembentukan citra *master share* menggunakan skema VSS (2, 2). Citra *master share* dibentuk berdasarkan matriks biner dari citra *host*. Setelah citra *master share* didapatkan, langkah selanjutnya adalah proses pembentukan *ownership share*. *Ownership share* dibentuk dari citra *watermark* dan citra *master share* menggunakan skema VSS (2, 2). Citra *ownership share* ini harus diautentikasi dengan CA untuk keamanan lebih lanjut. Penggabungan bermacam-macam metode yang digunakan ini bertujuan untuk meningkatkan kemampuan algoritma *blind watermarking* agar semakin *robust* dan lebih aman dari serangan *hacker*.

Penjelasan proses penyisipan *watermark* mulai dari pembentukan vektor fitur sampai pembentukan citra *ownership share* dijelaskan lebih detil dalam *pseudocode*. Terdapat satu *pseudocode* untuk program utama yang memanggil fungsi-fungsi yang digunakan dan beberapa *pseudocode* untuk masing-masing fungsi yang dipanggil.

3.3.1. Perancangan Program Utama Proses Penyisipan *Watermark*

Program utama merupakan program yang memanggil fungsi-fungsi lain untuk menjalankan program secara keseluruhan. Proses-proses yang dilakukan pada setiap tahap dipisahkan dalam fungsi-fungsi yang berbeda. Pemisahan ini bertujuan untuk memudahkan pengecekan kebenaran program untuk masing-masing proses. *Pseudocode* program utama ditunjukkan pada Gambar 3.8. Baris 1-10 menunjukkan proses pembentukan fitur citra. Sedangkan baris ke-11 dan ke-12 menunjukkan proses pembentukan *master share* dan *ownership share*. *Pseudocode* untuk masing-masing fungsi yang dipanggil dijelaskan lebih lanjut pada subbab berikutnya.

Masukan	host, secret, alpha, beta, A, B, type
Keluaran	m_owner

```
1.   matriks citra host dibagi ke dalam blok-blok non-
     overlapping sejumlah 64×64 menghasilkan
     block{1:64, 1:64}
2.   blockzigzag = zigzag(block);
3.   blockcatmap = catmap(blockzigzag, 64, A, B)
4.   for i=1 to 64
5.     for j=1 to 64
6.         blockfrft = frft2d(blockzigzag, [alpha
     beta]);
7.         m_singular(i,j)=svds(blockfrft{i,j},1)
8.     end for
9.   end for
10.  m_biner = binaryMap(m_singular)
11.  m_master = vss_master(m_biner, type)
12.  m_owner = vss_owner(secret, m_master)
```

Gambar 3.8 *Pseudocode* Program Utama

3.3.2. Pengacakan Piksel

Sebelum dilakukan pengacakan, citra *host* dibagi ke dalam blok-blok piksel sejumlah $m \times n$ (ukuran citra *watermark*). Proses pengacakan piksel dilakukan pada setiap blok. Sehingga yang berubah adalah posisi untuk masing-masing blok piksel. Terdapat dua fungsi yang digunakan untuk melakukan pengacakan, yaitu fungsi zigzag *scanning* dan fungsi *cat map*. Penggabungan dua fungsi ini bertujuan untuk mendapatkan posisi acak yang tidak mudah ditebak, sehingga sistem menjadi lebih aman. Penjelasan lebih lanjut untuk setiap fungsi tersebut disampaikan pada subbab berikut.

3.3.2.1. Perancangan Fungsi Zigzag *Scanning*

Fungsi zigzag *scanning* merupakan program yang melakukan pengacakan piksel dengan cara membaca blok-blok piksel secara zigzag dari kiri atas sampai kanan bawah. Hasil pembacaan zigzag disimpan ke dalam *cell* berukuran 1×4096.

Untuk membentuk kembali menjadi *cell* yang berukuran 64×64, maka dilakukan fungsi *reshape*. *Pseudocode* untuk proses zigzag *scanning* ditunjukkan pada Gambar 3.9 dan Gambar 3.10. Baris ke-1 dan ke-2 menunjukkan proses inisialisasi *cell* yang digunakan untuk menyimpan hasil pembacaan zigzag *scanning* dan inisialisasi posisi awal. Baris ke-4 sampai ke-40 menunjukkan proses penyimpanan blok yang dibaca secara zigzag ke dalam *cell* 1×4096. Baris ke-42 menunjukkan proses *reshape*.

Masukan	block
Keluaran	blockzigzag

```
1.  hsl_zigzag = cell(1,4096);
2.  cur_row = 1; cur_col=1; cur_index=1;
3.
4.  while (cur_row <= 64 && cur_col<= 64)
5.      if (cur_row == 1 && cur_col ~= 64 &&
        mod(cur_row+cur_col,2)== 0)
6.      hsl_zigzag{cur_index}=
        block{cur_row,cur_col};
7.      cur_col = cur_col+1;
8.      cur_index = cur_index+1;
9.
10.     elseif (cur_row == 64 && cur_col ~= 64 &&
        mod(cur_row+cur_col,2)~= 0)
11.     hsl_zigzag{cur_index}=
        block{cur_row,cur_col};
12.     cur_col = cur_col+1;
13.     cur_index = cur_index+1;
14.
15.     elseif (cur_row ~= 64 && cur_col == 1 &&
        mod(cur_row+cur_col,2)~= 0)
16.     hsl_zigzag{cur_index}=
        block{cur_row,cur_col};
17.     cur_row = cur_row+1;
18.     cur_index = cur_index+1;
19.
20.     elseif (cur_row ~= 64 && cur_col == 64 &&
        mod(cur_row+cur_col,2)== 0)
21.     hsl_zigzag{cur_index}=
        block{cur_row,cur_col};
```

Gambar 3.9 *Pseudocode* Fungsi Zigzag *Scanning* (Bagian 1)

```
22.     cur_row = cur_row+1;
23.     cur_index = cur_index+1;
24.
25.     elseif (cur_row ~= 64 && cur_col ~= 1 &&
        mod(cur_row+cur_col,2)~= 0)
26.     hsl_zigzag{cur_index}=
        block{cur_row,cur_col};
27.     cur_row = cur_row+1;
28.     cur_col = cur_col-1;
29.     cur_index = cur_index+1;
30.
31.     elseif (cur_row ~= 64 && cur_col ~= 1 &&
        mod(cur_row+cur_col,2)= 0)
32.     hsl_zigzag{cur_index}=
        block{cur_row,cur_col};
33.     cur_row = cur_row-1;
34.     cur_col = cur_col+1;
35.     cur_index = cur_index+1;

36.     elseif (cur_row == 64 && cur_col==64)
37.     hsl_zigzag{4096} = block{cur_row,cur_col};
38.     break
39.     end if
40. end while
41.
42. reshape_out = reshape(hsl_zigzag,64,64);
```

Gambar 3.10 *Pseudocode* Fungsi Zigzag *Scanning* (Bagian 2)

3.3.2.2. Perancangan Fungsi *Cat Map*

Fungsi *cat map* merupakan program yang melakukan pengacakan piksel dengan mengikuti Persamaan 2.25. Proses pengacakan *cat map* ini menggunakan masukan kunci A dan B. Keberadaan kedua kunci ini bertujuan untuk meningkatkan keamanan algoritma yang diusulkan. Sedangkan nilai variabel N yang digunakan bergantung pada ukuran citra *watermark*. Hasil program *cat map* berupa blok-blok piksel yang mengalami pergeseran ke dalam arah sumbu *y*, kemudian ke dalam arah sumbu

x. Semua hasilnya di modulo *N* agar hasilnya tetap pada area citra. *Pseudocode* untuk proses *cat map* ditunjukkan pada Gambar 3.11.

Masukan	blockzigzag, size_N, A, B
Keluaran	blockcatmap

```
1. out = cell(size_N, size_N)
2. for i=1 to size_N
3.    for j=1 to size_N
4.       index = mod([1 A; B A*B+1]*[i;j], size_N)+1
5.       blockcatmap{index(1), index(2)} =
   blockzigzag{i,j}
6.    endfor
7. endfor
```

Gambar 3.11 *Pseudocode* Fungsi *Cat Map*

3.3.3. Perancangan Fungsi Transformasi *frft2d*

Fungsi *frft2d* merupakan fungsi yang menjalankan transformasi 2D-DFRFT terhadap citra *host*. Proses transformasi dilakukan pada masing-masing blok piksel hasil pengacakan. Tujuannya adaah untuk meningkatkan ketahanan *watermark* terhadap gangguan yang bersifat geometris.

Masukan	host, alpha, beta
Keluaran	hslfrft

```
1.  [row_host, col_host] = size(host)
2.  for i = 1 to row_host
3.      temp(i,:) =
    dfrft1d(double(host(i,:)),alpha)
4.  end for
5.  for i = 1 to col_host
6.      hslfrft(:,i) = dfrft1d(temp(:,i),beta))
7.  end for
```

Gambar 3.12 *Pseudocode* Fungsi Transformasi Menggunakan 2D-DFRFT (*frft2d*)

Fungsi transformasi 2D-DFRFT sebenarnya adalah bentuk transformasi 1D-DFRFT pada baris dan kolom. Fungsi ini memiliki masukan berupa citra *host* dan dua parameter yaitu α dan β. Masing-masing kunci tersebut menentukan orde transformasi

yang dilakukan dan sensitif terhadap transformasi yang dihasilkan. Kunci α menentukan orde transformasi 1D-DFRFT pada vektor baris, sedangkan kunci β menentukan orde transformasi 1D-DFRFT pada vektor kolom. *Pseudocode* fungsi transformasi 2D-DFRFT ditunjukkan pada Gambar 3.12.

3.3.4. Perancangan Fungsi Transformasi d*frft1d*

Fungsi *dfrft1d* merupakan fungsi yang menjalankan transformasi 2D-DFRFT untuk masing-masing vektor baris dan kolom pada citra *host.* Fungsi ini memiliki masukan berupa vektor *f* yang menyatakan vektor baris atau kolom, *a* sebagai orde transformasi dan *p* sebagai penaksiran orde untuk pembentukan *eigenvector*. Nilai *p* bersifat *optional* dan diberi nilai default $N/2$, dengan *N* adalah panjang vektor *f*. Transformasi 1D-DFRFT terhadap vektor *f* dilakukan dengan bantuan transformasi kernel sesuai dengan Persamaan 2.7 dan 2.13. *Pseudocode* fungsi transformasi *dfrft1d* ditunjukkan pada Gambar 3.13.

Masukan	Vektor f, a, p
Keluaran	Vektor y

```
1.  Inisialisasi N = length(f), even = ~rem(N,2)
2.  shft = rem((0:N-1) + fix(N/2),N)+1
3.  f = f(:)
4.  if (nargin == 2), p = N/2
5.  end if
6.  p = min(max(2,p),N-1)
7.
8.  y(shft,1) = E*(exp(-j*pi/2*a*([0:N-2 N-
    1+even])).' .*(E'*f(shft)))
```

Gambar 3.13 *Pseudocode* Fungsi Transformasi Menggunakan 1D-DFRFT (*dfrft1d*)

Eigenvector yang digunakan untuk transformasi kernel didapatkan dengan mengikuti algoritma yang dijelaskan pada subbab 2.4.2. *Pseudocode* untuk proses ini ditunjukkan pada Gambar 3.14. Proses pembentukan *eigenvector E* bergantung pada panjang vektor *f* dan orde *p*.

Masukan	N, p
Keluaran	E

```
1.  inisialisasi d2 = [1 -2 1], d_p = 1, s = 0, st
    = zeros(1,N)
2.  for k = 1 to p/2
3.      d_p = conv(d2,d_p)
4.      st([N-k+1:N,1:k+1]) = d_p, st(1) = 0
5.      temp = [1,1:k-1;1,1:k-1]
6.      temp = temp(:)'./[1:2*k]
7.      s = s + (-1)^(k-1)*prod(temp)*2*st
8.  end for

9.  % S = circulant + diagonal
10. col = (0:N-1)'
11. row = (N:-1:1)
12. idx = col(:,ones(N,1)) + row(ones(N,1),:)
13. st = [s(N:-1:2).';s(:)]
14. S = st(idx) + diag(real(fft(s)))

15. % Construct transformation matrix P
16. Inisialisasi r = floor(N/2), even = ~rem(N,2)
17. P1 = (eye(N-1) + flipud(eye(N-1))) / sqrt(2)
18. P1(N-r:end,N-r:end) = -P1(N-r:end,N-r:end)
19. if (even), P1(r,r) = 1  end if
20. P = eye(N),  P(2:N,2:N) = P1

21. % Compute eigenvectors
22. PSP = P*S*P'
23. E = zeros(N)
24. Ev = PSP(1:r+1,1:r+1),  Od = PSP(r+2:N,r+2:N)
25. [ve,ee] = eig(Ev),     [vo,eo] = eig(Od)
26. E(1:r+1,1:r+1) = fliplr(ve)
27. E(r+2:N,r+2:N) = fliplr(vo)
28. E = P*E

29. % shuffle eigenvectors
30. ind = [1:r+1;r+2:2*r+2], ind = ind(:)
31. if (even), ind([N,N+2]) = []
32. else ind(N+1) = []  end if
33. E = E(:,ind')
```

Gambar 3.14 *Pseudocode* Fungsi Pembentukan Matriks E

Terdapat 5 tahapan yang dilalui untuk mendapatkan *eigenvector E*. Tahap pertama adalah pembentukan matriks *S*. Matriks S dibentuk dengan pendekatan orde *p* dengan melakukan penjumlahan matriks *circulant* dan matriks diagonal. *Pseudocode* pembentukan matriks *S* ditunjukkan pada baris kode ke-1 sampai ke-14. Tahap ke-2 adalah pembentukan matriks *P* yang ditunjukkan pada baris kode ke-15 sampai ke-20. Tahap ke-3 adalah dekomposisi matriks *even* dan *odd* dengan mengikuti Persamaan 2.10 yang ditunjukkan pada baris kode ke-24.

Tahap ke-4 adalah menentukan *eigenvector* dari matriks *even* dan *odd* dan mengurutkannya berdasarkan *eigenvalue* yang terbesar. *Pseudocode* untuk tahapan ini ditunjukkan pada baris kode ke-25 sampai ke-28. Tahap terakhir adalah membentuk matriks *eigenvector* terurut dengan tambahan matriks *zero* sesuai dengan Persamaan 2.11 dan Persamaan 2.12.

3.3.5. Perancangan Fungsi SVD

Fungsi SVD merupakan program yang melakukan dekomposisi nilai singular pada masing-masing blok piksel. Proses dekomposisi dijalankan dengan memanggil fungsi *svds* yang sudah disediakan oleh MATLAB. Fungsi *svds* mengembalikan *n* nilai singular pertama dari sebuah matriks. Hasil dekomposisi selanjutnya disimpan pada sebuah matriks, sehingga diperoleh matriks baru berukuran 64×64. *Pseudocode* fungsi dekomposisi nilai singular ditunjukkan oleh Gambar 3.15.

Masukan	blockcatmap
Keluaran	m_singular

```
1. for i=1 to 64
2.     for j=1 to 64
3.         m_singular(i,j)= svds(blockcatmap{i,j}, 1);
4.     end for
5. end for
```

Gambar 3.15 *Pseudocode* Fungsi SVD

3.3.6. Perancangan Fungsi Pembentukan Matriks Biner

Program pembentukan matriks biner adalah program yang menjalankan fungsi untuk mengubah matriks nilai singular menjadi matriks biner. Pemetaan ke dalam nilai biner dilakukan dengan mengikuti Persamaan 3.1.

$$B_{i,j} = \begin{cases} 0 \ \ if \ X_{i,j} < X_{av} \\ 1 \ \ if \ X_{i,j} \geq X_{av} \end{cases} \quad (3.1)$$

Di mana $X_{i,j}$ adalah elemen matriks biner pada baris ke-1 dan kolom ke-(i, j), X_{av} adalah rata-rata nilai matriks singular, 1 menotasikan piksel putih dan 0 menotasikan piksel hitam. *Pseudocode* pembentukan matriks biner ditunjukkan oleh Gambar 3.16.

Masukan	m_singular
Keluaran	m_biner

```
1.  avg = mean2(m_singular)
2.  [m,n]= size(m_singular)
3.  for i=1 to m
4.    for j=1 to n
5.      if (m_singular(i,j)< avg)
6.          m_biner(i,j)=0
7.      else
8.          m_biner(i,j)=1
9.      end if
10.   end for
11. end for
```

Gambar 3.16 *Pseudocode* Fungsi *binaryMap*

3.3.7. Perancangan Fungsi Pembentukan *Master Share* dengan VSS

Program pembentukan *master share* adalah program yang menjalankan fungsi untuk membentuk matriks biner yang berukuran $2m \times 2n$ piksel. Di mana $m \times n$ adalah ukuran matriks biner. Proses pembentukan *master share* mengikuti aturan yang

ditunjukkan pada Tabel 3.10. Proses pembentukan *master share* ini menggunakan masukan kunci *type* untuk menentukan jenis posisi VSS. *Type* =1 untuk diagonal, *type*=2 untuk horisontal dan *type*=3 untuk vertikal.

Di mana B_i adalah piksel pada matriks biner dan m_i adalah blok piksel pada *master share*. *Pseudocode* pembentukan *master share* ditunjukkan oleh Gambar 3.17 dan Gambar 3.18.

Tabel 3.10 Aturan Pembentukan *Master Share*

	Diagonal	*Horizontal*	*Vertical*
If B_i*=0*	$m_i = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$m_i = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	$m_i = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$
If B_i*=1*	$m_i = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$m_i = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$	$m_i = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$

Masukan	m_biner, type
Keluaran	m_master

```
1.  Inisialisasi [m,n] = size(m_biner), m_master =
    zeros(2*m, 2*n)
2.  switch type
3.      case 1
4.          for i=1 to m
5.              for j=1 to n
6.                  if (m_biner(i,j)==1)
7.                      m_master(2*i-1, 2*j-1)=1
8.                      m_master(2*i-1, 2*j)=0
9.                      m_master(2*i, 2*j-1)=0
10.                     m_master(2*i, 2*j)=1
11.                 else
12.                     m_master(2*i-1, 2*j-1)=0
13.                     m_master(2*i-1, 2*j)=1
14.                     m_master(2*i, 2*j-1)=1
15.                     m_master(2*i, 2*j)=0
16.                 end if
17.             end for
18.         end for
```

Gambar 3.17 *Pesudocode* Fungsi Pembentukan *Master Share* dengan VSS (Bagian 1)

```
19. case 2
20.         for i=1 to m
21.             for j=1 to n
22.                 if (m_biner(i,j)==1)
23.                     m_master(2*i-1, 2*j-1)=0
24.                     m_master(2*i-1, 2*j)=0
25.                     m_master(2*i, 2*j-1)=1
26.                     m_master(2*i, 2*j)=1
27.                 else
28.                     m_master(2*i-1, 2*j-1)=1
29.                     m_master(2*i-1, 2*j)=1
30.                     m_master(2*i, 2*j-1)=0
31.                     m_master(2*i, 2*j)=0
32.                 end if
33.             end for
34.         end for
35. case 3
36.         for i=1 to m
37.             for j=1 to n
38.                 if (m_biner(i,j)==1)
39.                     m_master(2*i-1, 2*j-1)=0
40.                     m_master(2*i-1, 2*j)=1
41.                     m_master(2*i, 2*j-1)=0
42.                     m_master(2*i, 2*j)=1
43.                 else
44.                     m_master(2*i-1, 2*j-1)=1
45.                     m_master(2*i-1, 2*j)=0
46.                     m_master(2*i, 2*j-1)=1
47.                     m_master(2*i, 2*j)=0
48.                 end if
49.             end for
50.         end for
51.     end switch
```

Gambar 3.18 *Pesudocode* Fungsi Pembentukan *Master Share* dengan VSS (Bagian 2)

Baris ke-3 sampai ke-18 menunjukkan *pseudocode* untuk posisi diagonal. Baris ke-19 sampai ke-34 menunjukkan *pseudocode* untuk posisi horisontal. Baris ke-35 sampai ke-51 menunjukkan *pseudocode* untuk posisi vertikal.

3.3.8. Perancangan Fungsi Pembentukan *Ownership Share* dengan VSS

Program pembentukan *ownership share* adalah program yang menjalankan proses penyisipan citra *watermark*. *Ownership share* dibentuk berdasarkan piksel-piksel pada citra *watermark* dan *master share* dengan mengikuti skema VSS (2, 2) pada Gambar 2.7. Jika piksel pada citra *watermark* bernilai 1 atau putih maka $o_i = m_i$. Sedangkan jika piksel pada citra *watermark* bernilai 0 atau hitam maka $o_i = \sim m_i$. Di mana o_i adalah blok piksel *ownership share* dan m_i adalah blok piksel *master share*. *Pseudocode* proses pembentukan *ownership share* ditunjukkan pada Gambar 3.19.

Masukan	secret, m_master
Keluaran	m_owner

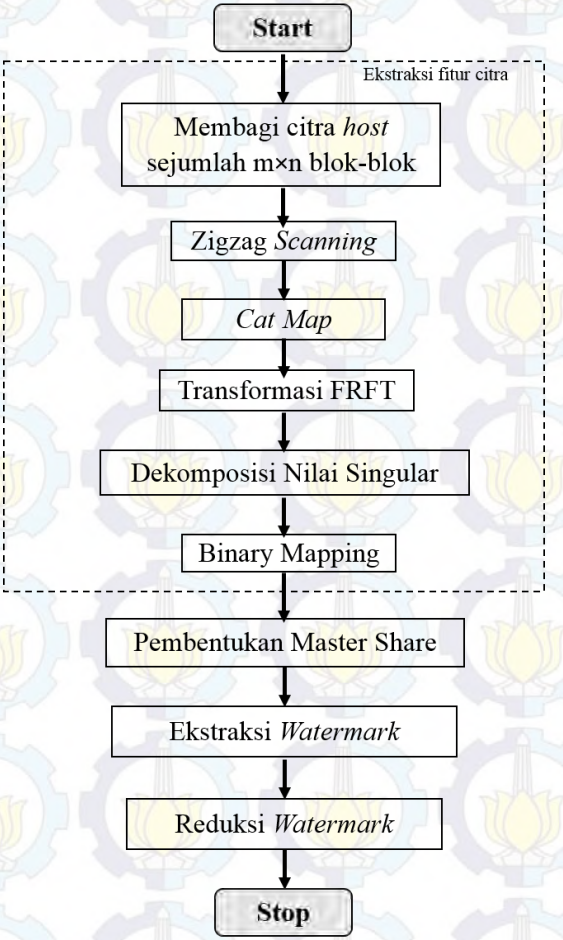
```
1.  [m,n] = size(secret)
2.  m_owner = m_master
3.  for i=1 to m
4.      for j=1 to n
5.          if (secret(i,j)==0)
6.              m_owner(2*i-1, 2*j-1)=
      ~(m_master(2*i-1, 2*j-1))
7.              m_owner(2*i-1, 2*j)= ~(m_master(2*i-
      1, 2*j))
8.              m_owner(2*i, 2*j-1)= ~(m_master(2*i,
      2*j-1))
9.              m_owner(2*i, 2*j)= ~(m_master(2*i,
      2*j))
10.         end if
11.     end for
12. end for
```

Gambar 3.19 *Pseudocode* Fungsi Pembentukan *Ownership Share* dengan VSS

3.4. Perancangan Proses Ekstraksi *Watermark*

Proses utama selanjutnya yang dibahas pada subbab ini adalah proses ekstraksi *watermark*. Diagram alir proses ekstraksi

watermark ditunjukkan pada Gambar 3.1 (b) pada subbab sebelumnya. Dapat dilihat masukan pada proses ekstraksi ada dua, yaitu citra yang diklaim berupa citra *grayscale* dan citra *ownership share* sebagai kunci ekstraksi yang menunjukkan kepemilikan *copyright*. Sedangkan tahap-tahap yang dilakukan pada proses ekstraksi *watermark* ditunjukkan pada Gambar 3.20.



Gambar 3.20 Diagram Alir Proses Ekstraksi *Watermark*

Tahap pertama proses ekstraksi *watermark* dimulai dengan pembentukan vektor fitur dari citra *host*. Tahapan pembentukan vektor fitur pada proses ekstraksi *watermark* sama persis dengan pembentukan vektor fitur pada proses penyisipan *watermark*. Perbedaannya hanya terletak pada masukan yang digunakan. Pada proses ekstraksi, vektor fitur diambil dari citra yang diklaim. Citra yang diklaim dapat berupa citra *host* yang belum mengalami gangguan maupun yang sudah mengalami gangguan.

Tahap selanjutnya adalah pembentukan *master share* dengan menggunakan masukan dari vektor fitur. Setelah didapatkan citra *master share*, langkah selanjutnya adalah ekstraksi *watermark* dengan cara menumpuk kedua *share*. Karena ukuran kedua *share* dua kali lebih besar dari citra *watermark*, maka hasil ekstraksi *watermark* (citra *stack*) pun juga berukuran lebih besar dari citra *watermark* yang disisipkan. Oleh sebab itu, perlu dilakukan proses reduksi citra *stack*.

Perancangan proses ekstraksi *watermark* mulai dari pembentukan vektor fitur sampai *master share* sampai proses reduksi citra *watermark* dijelaskan lebih detil dalam *pseudocode*. Terdapat satu *pseudocode* untuk program utama yang memanggil fungsi-fungsi yang digunakan dan beberapa *pseudocode* untuk masing-masing fungsi yang dipanggil.

3.4.1. Perancangan Program Utama Proses Ekstraksi *Watermark*

Program utama merupakan program yang memanggil fungsi-fungsi lain untuk menjalankan program secara keseluruhan. Proses-proses yang dilakukan dipisahkan dalam fungsi-fungsi yang berbeda untuk memudahkan pengecekan kebenaran program. *Pseudocode* program utama ditunjukkan pada Gambar 3.21. Baris kode ke-1 sampai ke-10 adalah *pseudocode* pemanggilan fungsi untuk pembentukan fitur citra *host*. Baris kode ke-11 adalah *pseudocode* pemanggilan fungsi pembentukan *master share*.

Sedangkan baris kode ke-12 dan ke-13 menunjukkan *pseudocode* pemanggilan fungsi untuk ekstraksi *watermark*.

Karena sebagian besar tahapan pada proses ekstraksi *watermark* sama persis dengan proses penyisipan *watermark*, maka pada subbab berikut hanya dijelaskan perancangan fungsi untuk ekstraksi *watermark* dan reduksi *watermark*. Sedangkan perancangan fungsi yang dipanggil pada proses pembentukan vektor fitur dan pembentukan *master share* dapat dilihat pada subbab 3.3.3 sampai subbab 3.3.7.

Masukan	hostAksen, m_owner, alpha, beta, type, A, B
Keluaran	reduceIMG

```
1.  matriks citra host dibagi ke dalam blok-blok
    non-overlapping sejumlah 64×64 menghasilkan
    block{1:64, 1:64}
2.  blockzigzag = zigzag(block);
3.  blockcatmap = catmap(blockzigzag, 64, A, B)
4.  for i=1 to 64
5.    for j=1 to 64
6.       blockfrft{i,j} = frft2d(blockzigzag,[alpha
    beta])
7.       m_singular(i,j)=svds(blockfrft{i,j},1)
8.    end for
9.  end for
10. m_biner = binaryMap(m_singular)
11. m_master = vssmaster(m_biner, type)
12. stackIMG = stacked(m_master, m_owner)
13. reduceIMG = reduced(stackIMG)
```

Gambar 3.21 *Pseudocode* Program Utama Proses Ekstraksi *Watermark*

3.4.2. Perancangan Fungsi Ekstraksi Citra *Watermark*

Fungsi ekstraksi citra *watermark* dirancang dengan cara menumpuk kedua *share*, yaitu *master share* dan *ownership share*. Proses ekstraksi citra *watermark* dilakukan dengan mengikuti skema VSS (2, 2) seperti yang ditunjukkan pada Gambar 2.7. Untuk menghasilkan citra *stack* yang sesuai dengan skema VSS (2, 2), maka dilakukan fungsi negasi xor untuk piksel yang berbeda.

Simulasi fungsi XOR dan proses ekstraksi ditunjukkan pada Tabel 3.11 dan Tabel 3.12. Sedangkan *pseudocode* proses ekstraksi citra *watermark* ditunjukkan pada Gambar 3.22.

Tabel 3.11 Simulasi Fungsi XOR

P	Q	P xor Q	P ~xor Q
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	1

Tabel 3.12 Simulasi Proses Ekstraksi

Master Share	*Ownership Share*	*Stack Result*
$\begin{bmatrix}1 & 0\\1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 0\\1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 0\\1 & 0\end{bmatrix}$
Jika piksel yang bersesuaian memiliki nilai yang sama, maka hasil *stack* sama dengan piksel *ownership share*.		
$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$
Jika piksel yang bersesuaian memiliki nilai yang berbeda, maka digunakan fungsi negasi xor (`~xor`) agar menghasilkan nilai 0.		

Masukan	m_master, m_owner
Keluaran	stackIMG

```
1.  [row_share,col_share]=size(m_owner);
2.  stackIMG = zeros(row_share,col_share)
3.
4.  for i=1 to row_share
5.     for j=1 to col_share
6.        if (m_master (i,j) == m_owner(i,j))
7.           secretS(i,j)= m_owner(i,j)
8.        else
9.           stackIMG(i,j)= ~xor(m_master (i,j),
    m_owner(i,j))
10.       end if
11.    end for
12. end for
```

Gambar 3.22 *Pseudocode* Fungsi Ekstraksi Citra *Watermark*

3.4.3. Perancangan Fungsi Reduksi Citra *Watermark* Hasil Ekstraksi

Fungsi reduksi bertujuan untuk mendapatkan citra *watermark* hasil ekstraksi yang lebih kecil seperti ukuran asli, sehingga dapat dilakukan uji NC terhadap citra *watermark* yang asli. Program ini diawali dengan membagi citra *watermark* hasil ekstraksi menjadi blok-blok berukuran 2×2 *non-overlapping*. Proses reduksi dilakukan dengan mengoperasikan masing-masing blok piksel dengan mengikuti Persamaan 3.2.

$$S''_{i,j} = \begin{cases} 0 \; if \sum_i \sum_j s'_{i,j} < 2 \\ 1 \; if \sum_i \sum_j s'_{i,j} \geq 2 \end{cases} \quad (3.2)$$

Masukan	stackIMG
Keluaran	reduceIMG

```
1.  [row_s,col_s]= size(stackIMG)
2.  reduceIMG=zeros(row_s/2,col_s/2);
3.  matriks citra stackIMG dibagi ke dalam blok-
    blok 2×2 menghasilkan block_stack{1:64, 1:64}
4.
5.  for i=1 to row_s
6.    for j= to col_s
7.      sum = jumlah nilai elemen block_stack{i,j}
8.      if (sum < 2)
9.          reduceIMG(i,j)=0
10.     else if(sum >= 2)
11.         reduceIMG(i,j)=1
12.     end if
13.   end for
14. end for
```

Gambar 3.23 *Pseudocode* Fungsi Reduksi Citra *Watermark* Hasil Ekstraksi

Di mana s'adalah notasi untuk masing-masing blok dan S'' merupakan citra hasil reduksi. *Pseudocode* reduksi citra *watermark* hasil ekstraksi ditunjukkan pada Gambar 3.23. Baris ke-3 menunjukkan proses pembagian piksel ke dalam blok-blok. Baris

ke-5 sampai ke-14 merupakan proses komputasi masing-masing piksel sesuai dengan Persamaan 3.2.

3.5. Perancangan Penilaian Kualitas Citra

Program penilaian kualitas citra terdiri atas dua, yaitu PSNR untuk uji kualitas citra *host* dan NC untuk uji kualitas citra *watermark*. Kedua proses pengujian ini diimplementasikan ke dalam dua program yang berbeda. *Pseudocode* untuk kedua program pengujian tersebut disajikan pada subbab berikut.

3.5.1. Perancangan Fungsi PSNR

Program untuk uji PSNR dilakukan dengan mengikuti Persamaan 2.27 dan 2.28. *Pseudocode* program uji PSNR ditunjukkan pada Gambar 3.24. Baris ke-1 sampai ke-3 menunjukkan proses inisialisasi. Baris ke-5 sampai ke-9 adalah proses untuk mendapatkan nilai MSE, sedangkan *pseudocode* penghitungan nilai PSNR ditunjukkan pada baris ke-10.

Masukan	H1, H2
Keluaran	psnr_val

```
1.  [row,col] = size(H1)
2.  % the size of the original image
3.  size_host = row*col
4.
5.  s=0;
6.  for i = 1 to size_host
7.      s = s+(H2(i) - H1(i))^2
8.  end for
9.  MSE = s/size_host;
10. psnr_val = 10*log10((255)^2/MSE)
```

Gambar 3.24 *Pseudocode* Fungsi untuk Uji PSNR

3.5.2. Perancangan Fungsi NC

Program untuk uji NC dilakukan dengan mengikuti Persamaan 2.29. *Pseudocode* program uji NC ditunjukkan pada Gambar 3.25. Baris ke-1 dan ke-2 menunjukkan proses inisialisasi

ukuran citra *watermark*. Baris ke-4 sampai ke-8 menunjukkan proses fungsi xor untuk masing-masing piksel citra hasil reduksi terhadap citra *watermark* asli. *Pseudocode* penghitungan nilai NC ditunjukkan pada baris ke-8.

Masukan	reducedIMG, secret
Keluaran	nc_value

```
1. [m, n] = size(reducedIMG)
2. size_host = m*n;
3.
4. s=0;
5. for i = 1 to size_secret
6.      x = xor(secret(i),reducedIMG(i))
7.      s = s+x;
8. end for
9. nc_value = (size_secret-s)/size_secret
```

Gambar 3.25 *Pseudocode* Fungsi untuk Uji NC

3.6. Perancangan Antar Muka

Perancangan antar muka aplikasi *blind watermarking* ini terdiri dari dua, yitu perancangan antar muka untuk proses penyisipan *watermark* dan antar muka untuk proses ekstraksi *watermark*. Kedua perancangan antar muka tersebut dijelaskan lebih lanjut pada subbab berikut.

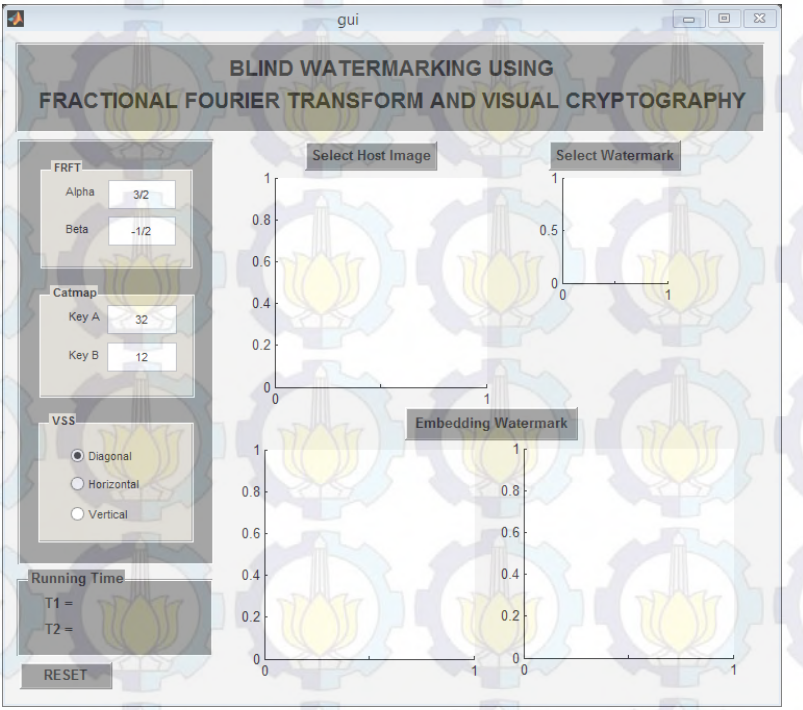
3.6.1. Perancangan Antar Muka Penyisipan *Watermark*

Pada antar muka program penyisipan *watermark* terdapat tiga tombol utama, yaitu tombol untuk memilih citra *host*, tombol untuk memilih citra *watermark* dan tombol untuk melakukan proses penyisipan. Pada sisi kiri terdapat sebuah panel yang berfungsi untuk memasukkan kunci yang digunakan pada proses penyisipan. Panel kunci sengaja diisi nilai *default* agar memudahkan pengguna.

Di bawah panel kunci juga terdapat satu panel lagi yang berfungsi untuk menampilkan total waktu running program penyisipan. Terdapat dua angka yang ditampilkan, yaitu T1 dan T2.

T1 adalah total waktu *running* program pembentukan *master share*. Sedangkan T2 adalah total waktu *running* program pembentukan *ownership share*. Tombol *reset* yang berada di kiri bawah digunakan untuk me-*reset* layar, sehingga aplikasi dapat digunakan lagi dengan masukan citra yang berbeda.

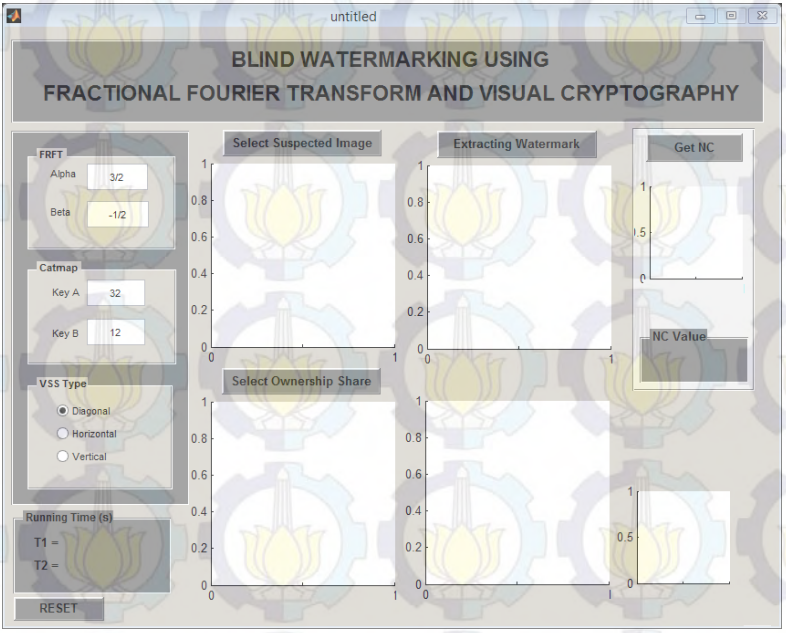
Hasil yang ditampilkan dari proses penyisipan *watermark* adalah citra *master share* dan citra *ownership share*. Di akhir proses, pengguna diminta untuk menyimpan citra *ownership share* dengan nama file dan direktori tertentu. Citra *ownership share* yang tersimpan selanjutnya dapat digunakan sebagai masukan pada program ekstraksi *watermark*. Rancangan antar muka untuk proses penyisipan *watermark* ditunjukkan oleh Gambar 3.26.



Gambar 3.26 Rancangan Antar Muka untuk Proses Penyisipan *Watermark*

3.6.2. Perancangan Antar Muka Ekstraksi *Watermark*

Pada antar muka program ekstraksi *watermark* terdapat empat tombol utama, yaitu tombol untuk memilih citra *host*, tombol untuk memilih citra *ownership share*, tombol untuk melakukan proses ekstraksi dan tombol untuk uji NC. Pada sisi kiri terdapat sebuah panel yang berfungsi untuk memasukkan kunci yang digunakan pada proses ekstraksi. Panel kunci sengaja diisi nilai *default* agar memudahkan pengguna.



Gambar 3.27 Rancangan Antar Muka untuk Proses Ekstraksi *Watermark*

Di bawah panel kunci juga terdapat satu panel lagi yang berfungsi untuk menampilkan total waktu running program ekstraksi. Terdapat dua angka yang ditampilkan, yaitu T1 dan T2. T1 adalah total waktu *running* program pembentukan *master share*

dari citra yang diklaim. Sedangkan T2 adalah total waktu *running* program untuk mengekstraksi citra *watermark*. Di bagian kiri bawah juga terdapat tombol *reset* yang dapat digunakan untuk me-*reset* layar.

Citra yang ditampilkan dari hasil proses ekstraksi *watermark* adalah citra *master share*, citra hasil ekstraksi, dan citra hasil ekstraksi yang sudah dikecilkan. Di bagian kanan tengah juga ditampilkan hasil uji NC terhadap citra *watermark* hasil ekstraksi. Rancangan antar muka untuk proses ekstraksi *watermark* ditunjukkan oleh Gambar 3.27.

BAB IV
IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Di awal bab akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1. Lingkungan Implementasi

Lingkungan yang digunakan untuk melakukan implementasi adalah program MATLAB 8.1.0.604 (R2013a) yang diinstal pada sistem operasi Windows 8.

4.2. Implementasi

Pada subbab ini dijelaskan implementasi dari setiap rancangan proses yang terdapat pada bab sebelumnya. Pada bagian implementasi ini juga dibagi menjadi dua bagian utama, yaitu implementasi proses penyisipan citra *watermark* dan implementasi proses ekstraksi citra *watermark*. Masing-masing proses diimplementasikan pada program utama yang berbeda dan memiliki fungsi-fungsi dan program pendukung. Pada bab ini juga disajikan penjelasan mengenai implementasi penilaian kualitas citra dan implementasi antar muka. Setiap bagian dijabarkan lebih lanjut pada subbab berikut.

4.2.1. Implementasi Proses Penyisipan Citra *Watermark*

Program utama proses penyisipan citra *watermark* diimplementasikan dalam *embeddGUI.m*. Program ini terdiri dari delapan tahap, yaitu :

1. Tahap inisialisasi, yaitu tahap untuk memasukkan citra *host*, citra *watermark* dan beberapa kunci (A, B, α, β dan posisi VSS).

2. Tahap pembagian citra *host* ke dalam ke dalam blok-blok piksel.
3. Tahap pengacakan posisi blok-blok piksel.
4. Tahap transformasi citra menggunakan *fractional Fourier transform* (2D-DFRFT).
5. Tahap dekomposisi nilai singular menggunakan SVD.
6. Tahap pembentukan matriks biner.
7. Tahap pembentukan *master share* menggunakan VSS.
8. Tahap pembentukan *ownership share* menggunakan VSS.

Masing-masing tahapan dijelaskan lebih detil pada subbab berikut.

4.2.1.1. Implementasi Program Utama Proses Penyisipan Citra *Watermark*

Tahap pertama dalam proses ini adalah inisialisasi di mana pengguna diminta untuk menentukan citra *host* dan citra *watermark*, menentukan parameter (A, B) untuk *cat map*, menentukan parameter (α, β) untuk FrFT dan menentukan posisi VSS. Implementasi tahap ini ditunjukkan pada Kode Sumber 4.1 sampai Kode Sumber 4.4.

```
1. [filename1,pathname1]=uigetfile('*.bmp;*.jpg;*.ti
   ff;*.png','Select the host image');
2. host=imread(num2str(filename1));

3. [filename2,pathname2]=uigetfile('*.bmp;*.jpg;*.ti
   ff;*.png','Select the secret image');
4. secret=imread(num2str(filename2));
```

Kode Sumber 4.1 Kode untuk Memilih dan Membaca Citra *Host* dan Citra *Watermark*

Tahap ke-2 adalah pembagian citra *host* ke dalam ke dalam blok-blok piksel sejumlah ukuran citra *watermark*. Proses pembagian blok dimulai dengan melakukan inisialisasi ukuran baris dan kolom untuk masing-masing blok serta jumlah blok yang

dibentuk. Proses inisialisasi ditunjukkan pada baris 2, 3 dan 5. Masing-masing blok hasil pembagian disimpan dalam sebuah *cell*. Implementasi pembagian dan penyimpanan blok-blok piksel ditunjukkan pada Kode Sumber 4.5.

```
1.  a_key = get(handles.edit1,'String');
2.  b_key = get(handles.edit2,'String');
3.
4.  ansAkey = strsplit(a_key,'/');
5.  if length(ansAkey)==1
6.        A = str2double(a_key);
7.  else A =
    str2double(ansAkey{1})/str2double(ansAkey{2});
8.  end;
9.  ansBkey = strsplit(b_key,'/');
10. if length(ansBkey)==1
11.       B = str2double(b_key);
12. else
13.       B =
    str2double(ansBkey{1})/str2double(ansBkey{2});
14. end;
```

Kode Sumber 4.2 Kode untuk Menentukan Kunci *Cat Map*

```
1.  alpha = get(handles.edit4,'String');
2.  beta = get(handles.edit3,'String');

3.  ansAlpha = strsplit(alpha,'/');
4.  if length(ansAlpha)==1
5.        alpha = str2double(alpha);
6.  else
7.        alpha =
    str2double(ansAlpha{1})/str2double(ansAlpha{2});
8.  end;
9.  ansBeta = strsplit(beta,'/');
10. if length(ansBeta)==1
11.       beta = str2double(beta);
12. else
13.       beta =
    str2double(ansBeta{1})/str2double(ansBeta{2});
14. end;
```

Kode Sumber 4.3 Kode untuk Menentukan Parameter 2D-DFRFT

```
type=handles.type;
type = str2double(type);
```

Kode Sumber 4.4 Kode untuk Menentukan Posisi VSS

```
% menentukan jumlah baris dan kolom blok piksel
row_block = floor(size(host,1)/size(secret,1));
col_block = floor(size(host,2)/size(secret,2));

% inisialisasi cell block sejumlah mxn
block = cell(m,n);

% membagi dan menyimpan citra host ke dalam blok
for i=1:m
    for j=1:n
        block{i,j}= hslfrft(i*row_block -
row_block+1 : i*row_block, j*col_block -
col_block+1 : j*col_block );
    end;
end;
```

Kode Sumber 4.5 Kode untuk Pembagian dan Penyimpanan Blok-Blok Piksel

Tahap ke-3 adalah pengacakan posisi blok-blok piksel dengan memanggil fungsi zigzag *scanning* (*zigzag.m*) dan fungsi *cat map* (*catmap.m*). Pemanggilan kedua fungsi ini ditunjukkan pada Kode Sumber 4.6.

```
% implementation of zigzag scanning
blockzigzag = zigzag(block);
% implementation of Cat map
size_N = n;
blockcatmap= catmap(blockzigzag, size_N,A, B);
```

Kode Sumber 4.6 Kode untuk Memanggil Fungsi *zigzag* dan *catmap*

Tahap ke-4 adalah implementasi transformasi citra *host* dengan memanggil fungsi *frft2d*. Sebelum dilakukan transformasi, blok-blok piksel citra *host* yang digunakan sebagai masukan diubah tipenya dari *uint8* ke tipe *double*. Karena hasil transformasi 2D-DFRFT berupa bilangan kompleks, maka koefisien hasil

transformasi diubah ke nilai absolut. Pemanggilan fungsi transformasi 2D-FRFT ditunjukkan pada Kode Sumber 4.7 baris ke-3 dan 4.

Tahap ke-5 adalah dekomposisi nilai singular pertama untuk masing-masing blok-blok piksel dengan memanggil fungsi *svds*. Pemanggilan fungsi ini ditunjukkan pada Kode Sumber 4.7 baris ke-5.

```
1. for i=1:m
2.     for j=1:n
3.         blockfrft{i,j}= frft2d(double(blockzigzag),
   [alpha beta]);
4.         blockfrft{i,j} = abs(blockfrft{I,j});
5.         m_singular(i,j)= svds(blockfrft{i,j}, 1);
6.     end;
7. end;
```

Kode Sumber 4.7 Kode untuk Memanggil Fungsi *frft2d* dan *svds*

Tahap ke-6 adalah pembentukan matriks biner dari matriks hasil dekomposisi nilai singular dengan memanggil fungsi *binaryMap* pada *binaryMap.m* yang akan dijelaskan pada subbab selanjutnya. Pemanggilan fungsi ini ditunjukkan pada Kode Sumber 4.8.

```
1. % construct binary map 64x64
2. m_biner = binaryMap(m_singular);
```

Kode Sumber 4.8 Kode untuk Memanggil Fungsi *binaryMap*

Tahap ke-7 adalah pembentukan citra *master share* dari citra *host* dengan memanggil fungsi *vssmaster* pada *vssmsater.m* yang akan dijelaskan pada subbab selanjutnya. Fungsi ini memiliki dua masukan, yaitu matriks biner dan tipe posisi VSS. Pemanggilan fungsi ini ditunjukkan pada Kode Sumber 4.9.

```
1. % generate master share with VSS position
2. m_master = vssmaster(m_biner,type);
```

Kode Sumber 4.9 Kode untuk Memanggil Fungsi *vssmaster*

Tahap ke-8 merupakan tahap pembentukan *ownership share* menggunakan VSS dengan memanggil fungsi *vssowner* pada *vssowner.m* yang akan dijelaskan pada subbab selanjutnya. Fungsi ini memiliki dua masukan, yaitu citra *watermark* dan *master share*. Pemanggilan fungsi ini ditunjukkan pada Kode Sumber 4.10.

```
1. % embedding watermark with dengan VSS
2. m_owner = vssowner (secret, m_master);
```

Kode Sumber 4.10 Kode untuk Memanggil Fungsi *vssowner*

4.2.1.2. Implementasi Fungsi Zigzag *Scanning*

Sebelum dilakukan proses zigzag *scanning*, citra *host* dibagi ke dalam blok-blok piksel. Selanjutnya blok-blok piksel tersebut dibaca secara zigzag dari kiri atas sampai kanan bawah. Hasil pembacaan kemudian disimpan ke dalam *cell* berukuran 1×4096. Untuk membentuk kembali menjadi *cell* yang berukuran 64×64, maka dilakukan fungsi *reshape*. Proses zigzag *scanning* dimaksudkan untuk mengubah posisi blok-blok piksel agar lebih acak dari sebelumnya.

Implementasi fungsi zigzag *scanning* ditunjukkan oleh Kode Sumber 4.11 sampai Kode Sumber 4.13. Pada baris 2 dan 3 dilakukan inisialisasi *cell* untuk menyimpan hasil pembacaan secara zigzag dan inisialisasi variabel untuk menentukan indeks posisi. Baris ke-5 sampai ke-41 menunjukkan proses pembacaan secara zigzag. Baris ke-43 menunjukkan proses mengembalikan hasil pembacaan ke bentuk *cell* 64×64.

```
1.  function [blockzigzag] = zigzag(block)
2.  hsl_zigzag = cell(1,4096);
3.  cur_row = 1; cur_col=1; cur_index=1;
4.
5.  while (cur_row <= 64 && cur_col<= 64)
6.       if (cur_row == 1 && cur_col ~= 64
```

Kode Sumber 4.11 Implementasi Fungsi *zigzag* (Bagian 1)

```
7.   &&mod(cur_row+cur_col,2)== 0)
8.           hsl_zigzag {cur_index}=
     block{cur_row,cur_col};
9.   cur_col = cur_col+1;
10.          cur_index = cur_index+1;
11.      elseif (cur_row == 64 && cur_col ~= 64 &&
     mod(cur_row+cur_col,2)~= 0)
12.          hsl_zigzag {cur_index}=
     block{cur_row,cur_col};
13.          cur_col = cur_col+1;
14.          cur_index = cur_index+1;
15.
16.      elseif (cur_row ~= 64 && cur_col == 1 &&
     mod(cur_row+cur_col,2)~= 0)
17.          hsl_zigzag {cur_index}=
     block{cur_row,cur_col};
18.          cur_row = cur_row+1;
19.          cur_index = cur_index+1;
20.
21.      elseif (cur_row ~= 64 && cur_col == 64 &&
     mod(cur_row+cur_col,2)== 0)
22.          hsl_zigzag {cur_index}=
     block{cur_row,cur_col};
23.          cur_row = cur_row+1;
24.          cur_index = cur_index+1;
25.
26.      elseif (cur_row ~= 64 && cur_col ~= 1 &&
     mod(cur_row+cur_col,2)~= 0)
27.          hsl_zigzag {cur_index}=
     block{cur_row,cur_col};
28.          cur_row = cur_row+1;
29.          cur_col = cur_col-1;
30.          cur_index = cur_index+1;
31.
32.      elseif (cur_row ~= 1 && cur_col ~=64 &&
     mod(cur_row+cur_col,2)== 0)
33.          hsl_zigzag {cur_index}=
     block{cur_row,cur_col};
34.          cur_row = cur_row-1;
35.          cur_col = cur_col+1;
36.          cur_index = cur_index+1;
37.
```

Kode Sumber 4.12 Implementasi Fungsi *zigzag* (Bagian 2)

```
38.     elseif (cur_row == 64 && cur_col==64)
39.         hsl_zigzag {4096} =
    block{cur_row,cur_col};
40.     break
41.     end;
42. end
43. % mengembalikan ke bentuk cell 64×64
44. blockzigzag = reshape(hsl_zigzag,64,64);
45. end
46.
```

Kode Sumber 4.13 Implementasi Fungsi *zigzag* (Bagian 3)

4.2.1.3. Implementasi Fungsi *Cat Map*

Proses pengacakan blok piksel menggunakan *cat map* dilakukan setelah proses zigzag *scanning*. Fungsi *cat map* diimplementasikan sesuai dengan Persamaan 2.19. Proses pengacakan *cat map* ini menggunakan masukan *N*, *A* dan *B*. Keberadaan kunci *A* dan *B* ini menentukan koordinat baru dari masing-masing blok. Sedangkan nilai *N* menunjukkan ukuran matriks yang akan dikomputasi dan nilainya disesuaikan dengan ukuran baris atau kolom citra *watermark*.

```
1.  function [ blockcatmap ] = catmap( blockzigzag,
    size_N,A,B)
2.  blockcatmap = cell(size_N,size_N);
3.  for i=1:size_N
4.      for j=1:size_N
5.          index = mod([1 A; B
    A*B+1]*[i;j],size_N)+1;
6.          blockcatmap{index(1), index(2)} =
    blockzigzag{i,j};
7.      end
8.  end
9.  end
```

Kode Sumber 4.14 Implementasi Fungsi *catmap*

Hasil program *cat map* berupa blok-blok piksel yang mengalami pergeseran ke dalam arah sumbu *y*, kemudian ke dalam arah sumbu *x*. Semua hasilnya di modulo *N* agar hasilnya tetap

pada area citra. Implementasi fungsi pengacakan *cat map* ditunjukkan pada Kode Sumber 4.14.

4.2.1.4. Implementasi Fungsi 2D-DFRFT

Fungsi transformasi citra menggunakan 2D-DFRFT dilakukan pada citra *host* yang berupa citra *grayscale*. Fungsi ini digunakan untuk mengubah citra ke dalam domain frekuensi.

Fungsi transformasi 2D-DFRFT sebenarnya adalah bentuk transformasi 1D-DFRFT pada baris dan kolom. Fungsi ini memiliki masukan berupa matriks dua dimensi dan dua buah kunci yang dibaca sebagai vektor, yaitu α pada indeks ke-1 dan β pada indeks ke-2. Masing-masing kunci tersebut menentukan orde transformasi yang dilakukan dan sensitif terhadap transformasi yang dihasilkan. Kunci α menentukan orde transformasi 1D-DFRFT pada setiap baris, sedangkan kunci β menentukan orde transformasi 1D-DFRFT pada setiap kolom.

```
1.  function [out] = frft2d(matrix,angles)
2.  for i = 1:size(matrix,1)
3.      temp(i,:) =
    dfrft1d(double(matrix(i,:)),angles(1));
4.  end
5.  for i = 1:size(matrix,2)
6.      out(:,i) = dfrft1d(temp(:,i),angles(2));
7.  end
```

Kode Sumber 4.15 Implementasi Fungsi 2D-DFRFT

Implementasi fungsi *frft2d.m* ditunjukkan oleh Kode Sumber 4.15. Baris kode 2-4 menunjukkan implementasi transformasi 1D-DFRFT pada setiap baris. Sedangkan baris kode 5-7 menunjukkan implementasi transformasi 1D-DFRFT pada setiap baris.

4.2.1.5. Implementasi Fungsi 1D-DFRFT

Fungsi transformasi 1D-DFRFT merupakan bentuk generalisasi dari fungsi transformasi DFT. Sehingga proses

pembentukan fungsi 1D-DFRFT tidak dapat dilepaskan dari matriks DFT. Algoritma pembentukan fungsi 1D-DFRFT didefinisikan menggunakan transformasi kernel dengan mengikuti Persamaan 2.7 dan 2.13. Implementasi fungsi 1D-DFRFT dengan transformasi kernel ditunjukkan oleh Kode Sumber 4.16 dan Kode Sumber 4.17. Implementasi Persamaan 2.13 ditunjukkan pada baris ke-9.

Matriks kernel DFRFT dibentuk berdasarkan *eigenvector* dan *eigenvalue* dari matriks DFT (matriks *U*). Pada implementasinya, matriks *U* dinotasikan dengan variabel E. Parameter *a* yang digunakan dalam fungsi ini menunjukkan pangkat dari *eigenvalue* (λ_k) matriks DFT. Karena matriks DFT hanya memiliki empat eigenvalue ($e^{-j(\pi/2)k} \in \{1, -1, -j, j\}$), maka *real eigenvector* (E) dari matriks kernel DFT didapatkan dengan menghitung *eigenvector* dari operasi komutatif matriks S dan F ($SF = FS$). Sehingga proses pembentukan eigenvector dilakukan menggunakan algoritma terpisah.

```
1.  function y = dfrft1d(f,a,p)
2.
3.  N = length(f); even = ~rem(N,2);
4.  shft = rem((0:N-1) + fix(N/2),N)+1;
5.  f = f(:);
6.  if (nargin == 2), p = N/2; end;
7.  p = min(max(2,p),N-1);
8.  E = dFRFT(N,p);
9.  y(shft,1) = E*(exp(-j*pi/2*a*([0:N-2 N-
    1+even])).' .*(E'*f(shft)));
10.
11. %---------------------------------------
12. function E = dFRFT(N,p)
13. global E_saved p_saved
14. if (length(E_saved) ~= N | p_saved ~= p),
15. E = make_E(N,p);
16. E_saved = E; p_saved = p;
17. else
18. E = E_saved;
19. end;
20. %---------------------------------------
```

Kode Sumber 4.16 Implementasi Fungsi *dfrft1d* (Bagian 1)

```
21. function E = make_E(N,p)
22. d2 = [1 -2 1]; d_p = 1; s = 0; st = zeros(1,N);
23. for k = 1:p/2,
24.     d_p = conv(d2,d_p);
25.     st([N-k+1:N,1:k+1]) = d_p; st(1) = 0;
26.     temp = [1,1:k-1;1,1:k-1];
27.     temp = temp(:)'./[1:2*k];
28.     s = s + (-1)^(k-1)*prod(temp)*2*st;
29. end;
30. % S = circulant + diagonal
31. col = (0:N-1)'; row = (N:-1:1);
32. idx = col(:,ones(N,1)) + row(ones(N,1),:);
33. st = [s(N:-1:2).';s(:)];
34. S = st(idx) + diag(real(fft(s)));
35.
36. % Construct transformation matrix P
37. r = floor(N/2);
38. even = ~rem(N,2);
39. P1 = (eye(N-1) + flipud(eye(N-1))) / sqrt(2);
40. P1(N-r:end,N-r:end) = -P1(N-r:end,N-r:end);
41. if (even), P1(r,r) = 1; end
42. P = eye(N); P(2:N,2:N) = P1;
43.
44. % Compute eigenvectors
45. PSP = P*S*P';
46. E = zeros(N);
47. Ev = PSP(1:r+1,1:r+1);
48. Od = PSP(r+2:N,r+2:N);
49. [ve,ee] = eig(Ev);
50. [vo,eo] = eig(Od);
51. E(1:r+1,1:r+1) = fliplr(ve);
52. E(r+2:N,r+2:N) = fliplr(vo);
53. E = P*E;
54.
55. % shuffle eigenvectors
56. ind = [1:r+1;r+2:2*r+2]; ind = ind(:);
57. if (even), ind([N,N+2]) = [];
58. else ind(N+1) = []; end
59. E = E(:,ind');
```

Kode Sumber 4.17 Implementasi Fungsi *dfrft1d* (Bagian 2)

Fungsi pembentukan *eigenvector E* diimplementasikan dengan mengikuti algoritma yang sudah dijelaskan pada subbab 2.4.2. Baris kode ke-21 sampai ke-34 menunjukkan implementasi pembentukan matriks *S* dengan pendekatan orde *p*. Orde *p* ditaksir dengan nilai minimal *p*=2 dan *default p*=*N*/2. Baris kode ke-37 sampai ke-42 menunjukkan implementasi pembentukan matriks *P*. Implementasi proses dekomposisi *eigenvector* dari matriks *even* dan *odd* ditunjukkan pada baris kode ke-45 sampai 50. Baris ke- 51 sampai ke-53 adalah implementasi Persamaan 2.11 dan 2.12. Sedangkan di bagian akhir program terdapat proses untuk mengubah susunan posisi *eigenvector* yang ditunjukkan pada baris ke-56 sampai 59.

4.2.1.6. Implementasi Fungsi SVD

Tujuan penggunaan SVD pada tugas akhir ini adalah untuk mendapatkan nilai singular pertama dari masing-masing blok piksel. Proses dekomposisi dijalankan dengan memanggil fungsi *svds* yang sudah disediakan oleh MATLAB. Fungsi *svds* mengembalikan *n* nilai singular pertama dari sebuah matriks. Karena yang dibutuhkan hanya satu nilai terbesar, maka *n* bernilai 1. Hasil dekomposisi selanjutnya disimpan pada sebuah matriks baru, sehinga diperoleh matriks baru berukuran 64×64 yang berisi nilai singular terbesar dari masing-masing blok. Implementasi fungsi SVD ditunjukkan pada Kode Sumber 4.18.

```
1. for i=1:m
2.     for j=1:n
3.         m_singular(i,j)= svds(blockfrft{i,j}, 1);
4.     end;
5. end;
```

Kode Sumber 4.18 Implementasi Fungsi SVD

4.2.1.7. Implementasi Fungsi Pembentukan Matriks Biner

Pembentukan mastriks biner diimplementasikan ke dalam sebuah fungsi *binaryMap.m*. Pada fungsi ini matriks singular dipetakan ke dalam piksel hitam putih (0, 1) dengan batas ambang berupa nilai rata-rata matriks singular sesuai dengan Persamaan 3.1. Implementasi fungsi ditunjukkan oleh Kode Sumber 4.19. Baris ke-3 menunjukkan proses untuk mendapatkan nilai rata-rata dari matriks singular. Sedangkan pemetaan ke dalam piksel 0 dan 1 ditunjukkan pada baris ke-4 sampai ke-12.

```
1.  function [ m_biner ] = binaryMap( m_singular )
2.  [m,n]=size(m_singular);
3.  avg= mean2(m_singular)      % compute average
    value of matriks singular
4.  for i=1:m
5.      for j=1:n
6.          if(m_singular(i,j)< avg)
7.              m_biner(i,j)=0;
8.          else
9.              m_biner(i,j)=1;
10.         end;
11.     end;
12. end;
13. end
```

Kode Sumber 4.19 Implementasi Fungsi Pembentukan Matriks Biner

4.2.1.8. Implementasi Fungsi VSS untuk Pembentukan *Master Share* (*vssmaster*)

Pembentukan *master share* diimplementasikan ke dalam sebuah fungsi *vssmaster.m*. Implementasi fungsi ini mengikuti aturan pembentukan *master share* pada Tabel 3.10. Masukan yang digunakan pada fungsi ini ada dua, yaitu matriks biner dan posisi VSS (diagonal, horisontal dan vertikal). Posisi VSS berfungsi untuk menentukan kombinasi posisi piksel hitam dan putih untuk masing-masing blok.

Hasil proses ini berupa matriks biner acak yang memiliki ukuran dua kali lebih besar dari ukuran matriks biner. Hal ini terjadi karena pada proses pembentukan *master share* masing-masing piksel pada matriks biner diganti dengan blok-blok piksel berukuran 2×2. Implementasi fungsi pembentukan *master share* ditunjukkan pada Kode Sumber 4.20 dan Kode Sumber 4.21. Baris ke-7 sampai ke-21 menunjukkan implementasi pembentukan *master share* dengan posisi diagonal, baris ke-24 sampai ke-38 untuk posisi horisontal dan baris ke-41 sampai ke-55 untuk posisi vertikal.

```
1. function [ m_master ] = VSS( m_biner, type )
2. [m,n] = size(m_biner);
3. m_master = zeros(2*m, 2*n);
4.  switch type
5.      case 1
6.          disp('VSS 1 Diagonal');
7.          for i=1:m
8.              for j=1:n
9.                  if (m_biner(i,j)==1)
10.                     m_master(2*i-1, 2*j-1)=1;
11.                     m_master(2*i-1, 2*j)=0;
12.                     m_master(2*i, 2*j-1)=0;
13.                     m_master(2*i, 2*j)=1;
14.                 else
15.                     m_master(2*i-1, 2*j-1)=0;
16.                     m_master(2*i-1, 2*j)=1;
17.                     m_master(2*i, 2*j-1)=1;
18.                     m_master(2*i, 2*j)=0;
19.                 end;
20.             end;
21.         end;
22.     case 2
23.         disp('VSS 2 Horizontal');
24.         for i=1:m
25.             for j=1:n
26.                 if (m_biner(i,j)==1)
27.                     m_master(2*i-1, 2*j-1)=0;
```

Kode Sumber 4.20 Implementasi Fungsi VSS untuk Pembentukan *Master Share* (Bagian 1)

```
28.                     m_master(2*i-1, 2*j)=0;
29.                     m_master(2*i, 2*j-1)=1;
30.                     m_master(2*i, 2*j)=1
31.                 else
32.                     m_master(2*i-1, 2*j-1)=1;
33.                     m_master(2*i-1, 2*j)=1;
34. m_master(2*i, 2*j-1)=0;
35.                     m_master(2*i, 2*j)=0;
36.                 end;
37.             end;
38.         end;
39.     case 3
40.         disp('VSS 3 Vertikal');
41.         for i=1:m
42.             for j=1:n
43.                 if (m_biner(i,j)==1)
44.                     m_master(2*i-1, 2*j-1)=0;
45.                     m_master(2*i-1, 2*j)=1;
46.                     m_master(2*i, 2*j-1)=0;
47.                     m_master(2*i, 2*j)=1;
48.                 else
49.                     m_master(2*i-1, 2*j-1)=1;
50.                     m_master(2*i-1, 2*j)=0;
51.                     m_master(2*i, 2*j-1)=1;
52.                     m_master(2*i, 2*j)=0;
53.                 end;
54.             end;
55.         end;
56.     End
```

Kode Sumber 4.21 Implementasi Fungsi VSS untuk Pembentukan *Master Share* (Bagian 2)

4.2.1.9. Implementasi Fungsi VSS untuk Pembentukan *Ownership Share* (*vssowner*)

Pembentukan *ownership share* diimplementasikan ke dalam sebuah fungsi *vssowner.m*. Implementasi fungsi ini mengikuti skema VSS sesuai dengan Gambar 2.7. Masukan yang digunakan pada fungsi ini ada dua, yaitu *master share* dan citra *watermark*. Jika piksel pada citra *watermark* bernilai 1 atau putih

maka $o_i = m_i$. Sedangkan jika piksel pada citra *watermark* bernilai 0 atau hitam maka $o_i = \sim m_i$. Di mana o_i adalah piksel *ownership share* dan m_i adalah piksel *master share*. Hasil proses ini berupa matriks biner acak yang memiliki ukuran yang sama dengan *master share*.

```
1.  function [ m_owner ] = ownershipShare( secret,
    m_master )
2.
3.  [m,n] = size(secret);
4.  m_owner = m_master;
5.
6.  for i=1:m
7.    for j=1:n
8.        if (secret(i,j)==0)
9.            m_owner(2*i-1, 2*j-1)=~(m_master(2*i-
    1, 2*j-1));
10.           m_owner(2*i-1, 2*j)=~(m_master(2*i-1,
    2*j));
11.           m_owner(2*i, 2*j-1)=~(m_master(2*i,
    2*j-1));
12.           m_owner(2*i, 2*j)=~(m_master(2*i,
    2*j));
13.       end;
14.     end;
15. end;
16. end
```

Kode Sumber 4.22 Implementasi Fungsi VSS untuk Pembentukan *Ownership Share* (*vssowner*)

Implementasi fungsi pembentukan *ownership share* ditunjukkan pada Kode Sumber 4.22. Baris kode ke-4 menunjukkan proses inisialisasi matriks *ownership share*. Baris kode 6-15 menunjukkan proses pembentukan matriks *ownership share* jika piksel pada citra *watermark* bernilai 0 atau hitam. Sebaliknya, jika citra *watermark* yang bernilai bernilai 1 atau putih, maka matriks *ownership share* dibiarkan saja.

4.2.2. Implementasi Proses Ekstraksi Citra *Watermark*

Program utama proses ekstraksi citra *watermark* diimplementasikan dalam *extractGUI.m*. Program ini terdiri dari sembilan tahapan, yaitu

1. Tahap inisialisasi, yaitu tahap memasukkan citra yang diklaim, citra *ownership share* dan beberapa kunci (A, B, α, β dan posisi VSS).
2. Tahap pembagian citra *host* ke dalam ke dalam blok-blok piksel.
3. Tahap pengacakan blok-blok piksel.
4. Tahap transformasi citra menggunakan 2D-DFRFT.
5. Tahap dekomposisi nilai singular menggunakan SVD.
6. Tahap pembentukan matriks biner.
7. Tahap pembentukan *master share* menggunakan VSS
8. Tahap ekstraksi citra *watermark* menggunakan VSS.
9. Tahap reduksi citra *watermark* hasil ekstraksi.

Penjelasan mengenai implementasi tahapan di atas disajikan pada subbab berikut.

4.2.2.1. Implementasi Program Utama Proses Ekstraksi Citra *Watermark*

Tahap pertama dalam proses ini adalah tahap inisialisasi di mana pengguna diminta untuk menentukan citra yang diklaim dan citra *ownership share*, menentukan parameter (A, B) untuk *cat map*, menentukan parameter (α, β) untuk FrFT dan menentukan posisi VSS. Implementasi program untuk menentukan citra masukan ditunjukkan pada Kode Sumber 4.23. Sedangkan implementasi program untuk menentukan kunci A, B, α, β dan posisi VSS dapat dilihat pada Kode Sumber 4.2 sampai Kode Sumber 4.4.

Sebagian besar tahapan proses ekstraksi sama dengan tahapan pada proses penyisipan, yaitu mulai dari pembagian citra *host* ke dalam blok-blok piksel sampai tahap pembentukan *master*

share. Masing-masing implementasi tahapan tersebut dapat dilihat pada subbab 4.2.1.1 sampai subbab 4.2.1.7. Sehingga pada subbab selanjutnya hanya dijelaskan implementasi tahap ekstraksi dan tahap reduksi citra *watermark* hasil ekstraksi.

```
1. [filename1,pathname1]=uigetfile('*.bmp;*.jpg;*.tiff;*.png','Select the suspected image');
2. host=imread(num2str(filename1));
3. [filename2,pathname2]=uigetfile('*.bmp;*.jpg;*.tiff;*.png','Select the ownership share');
4. m_owner =imread(num2str(filename2));
```

Kode Sumber 4.23 Kode untuk Memilih dan Membaca Citra *Host* dan Citra *Ownership Share*

4.2.2.2. Implementasi Fungsi VSS untuk Ekstraksi Citra *Watermark*

Program yang menjalankan proses ekstraksi citra *watermark* diimplementasikan ke dalam fungsi *stacked.m.* Fungsi ini diimplementasikan dengan cara menumpuk kedua *share*, yaitu *master share* dan *ownership share* sesuai dengan skema VSS (2, 2) seperti yang ditunjukkan pada Gambar 2.7.

```
1.  function [ stackIMG ] = stacked( m_master, m_owner )
2.  [row_share,col_share]=size(m_owner);
3.  stackIMG = zeros(row_share,col_share);
4.
5.  for i=1: row_share
6.    for j=1: col_share
7.       if (m_master(i,j) ~= m_owner(i,j))
8.           stackIMG (i,j)= ~xor(m_master(i,j), m_owner(i,j));
9.       else stackIMG (i,j)= m_owner(i,j);
10.      end;
11.   end;
12. end;
```

Kode Sumber 4.24 Implementasi Fungsi *stacked* untuk Ekstraksi Citra *Watermark*

Untuk menghasilkan citra *stack* yang sesuai dengan skema VSS (2, 2), maka dilakukan fungsi negasi *xor* untuk piksel yang berbeda. Simulasi fungsi *xor* dan proses ekstraksi ditunjukkan pada Tabel 3.11 dan Tabel 3.12. Sedangkan implementasi fungsi ekstraksi *watermark* ditunjukkan oleh Kode Sumber 4.24. Baris ke-2 dan ke-3 menunjukkan proses inisialisasi matriks hasil ekstraksi yang berukuran sama dengan *ownership share*. Sedangkan baris ke-5 sampai ke-12 menunjukkan proses komputasi untuk mengekstraksi *watermark* yang dilakukan pada masing-masing piksel.

4.2.2.3. Implementasi Fungsi Reduksi Citra Hasil Ekstraksi

Program yang menjalankan proses reduksi citra hasil ekstraksi diimplementasikan ke dalam fungsi *reduced.m*. Fungsi ini bertujuan untuk mendapatkan citra *watermark* hasil ekstraksi yang lebih kecil seperti ukuran asli. Sehingga dapat dilakukan uji NC terhadap citra *watermark* yang asli.

```
1.  function [ reduceIMG] = reduced( stackIMG )
2.
3.  [row_s,col_s] = size(stackIMG);
4.  row_block = 2;
5.  col_block = 2;

6.  % inisialisasi cell block
7.  blocks = cell(floor(row_s m/2), floor(col_s
    /2));
8.  % membagi citra ke dalam blok-blok non-
    overlapping
9.  for i=1: row_s /2
10.    for j=1: col_s /2
11.       blockSS{i,j}= stackIMG(i*row_block -
    row_block+1 : i*row_block, j*col_block -
    col_block+1 : j*col_block );
12.    end;
13. end;
```

Kode Sumber 4.25 Implementasi Fungsi Reduksi Citra *Watermark* Hasil Ekstraksi (Bagian 1)

```
14. reduceIMG=zeros(m/2,n/2);
15. for i=1:64
16.     for j=1:64
17. % compute total value of each blok
18.         sum = fsum(blockSS{i,j});
19.         if (sum < 2)
20.             reduceIMG(i,j)=0;
21.         else if(sum >= 2)
22.             reduceIMG(i,j)=1;
23.         end;
24.     end
25. end
26. end
27.
28. function [ sum ] = fsum( blockSS )
29. [m,n]=size(blockSS);sum=0;
30. for i=1:m
31.     for j=1:n
32. sum=sum+ blockSS(i,j);
33.     end
34. end
35. end
```

Kode Sumber 4.26 Implementasi Fungsi Reduksi Citra *Watermark* Hasil Ekstraksi (Bagian 2)

Proses reduksi diawali dengan pembagian piksel citra *stack* ke dalam blok-blok berukuran 2×2. Selanjutnya dilakukan dengan komputasi untuk masing-masing blok piksel mengikuti Persamaan 3.2.

Implementasi fungsi *reduced.m* dapat dilihat pada Kode Sumber 4.25 dan Kode Sumber 4.26 . Proses pembagian piksel ke dalam blok-blok berukuran 2×2 ditunjukkan pada barsi kode ke-9 sampai baris kode ke-13. Sedangkan proses komputasi untuk masing-masing blok ditunjukkan pada baris kode ke-15 sampai baris kode ke-27. Pada program ini terdapat sebuah subfungsi yang digunakan untuk menghitung jumlah nilai dalam satu blok piksel. Implementasi subfungsi ini ditunjukkan pada baris kode ke-28 sampai baris kode ke-35.

4.2.3. Implementasi Penilaian kualitas Citra

Implementasi penilaian kualitas citra dibuat ke dalam dua fungsi, yaitu *psnr.m* untuk uji PSNR dan *nc.m* untuk uji NC. Fungsi uji PSNR dipanggil diluar program utama. Fungsi ini dipanggil secara manual untuk masing-masing citra *host* yang sudah mengalami gangguan. Sedangkan fungi uji NC dipanggil setelah dilakukan fungsi ekstraksi citra *watermark*.

4.2.3.1. Implementasi Uji PSNR

Fungsi uji PSNR diimplementasikan sesuai dengan Persaman 2.27 dan 2.28. Fungsi ini memiliki dua masukan, yaitu citra asli dan citra yang sudah mengalami gangguan. Hasil yang dikembalikan fungsi ini menunjukkan kualitas citra dalam satuan desibel (dB). Implementasi fungsi uji PSNR ditunjukkan pada Kode Sumber 4.27.

```
1.  function [ psnr_value ] = PSNR( host,
    host_aksen )
2.  [row,col] = size(host); size_host = row*col;
3.  H1 = double(image1);
4.  H2 = double(image2);
5.  s=0;
6.  for i = 1:size_host
7.       s = s+(H2(i) - H1(i))^2 ;
8.  end
9.  MSE = s/size_host;
10. psnr_value = 10*log10((255)^2/MSE);
11. end
```

Kode Sumber 4.27 Implementasi Fungsi *psnr* untuk Uji PSNR

4.2.3.2. Implementasi Uji NC

Fungsi uji NC diimplementasikan mengikuti Persamaan 2.29. Fungsi ini memiliki dua masukan, yaitu citra *watermark* asli dan citra hasil ekstraksi. Hasil yang dikembalikan fungsi ini menunjukkan akurasi kemiripan terhadap citra asli dalam rentang

nilai 0-1. Implementasi fungsi uji NC ditunjukkan pada Kode Sumber 4.28.

```
1.  function [ out_NC ] = NC( secret, reducedIMG )
2.  [row,col] = size(secret);
3.  size_secret= row*col;
4.  secret1 = double(secret);
5.  secret2 = double(reducedIMG);
6.  s=0;
7.  for i = 1:size_secret
8.      x = xor(secret1(i),secret2(i));
9.      s = s+x:
10. end
11. out_NC = (size_secret-s)/size_secret;
12. end
```

Kode Sumber 4.28 Implementasi Fungsi *nc* untuk Uji NC

4.2.4. Implementasi Antar Muka

Implementasi antar muka dilakukan dengan membuat beberapa tombol yang berfungsi untuk memanggil setiap subproses yang ada dalam proses utama. Selain itu antar muka juga diimplementasikan untuk menampilkan citra masukan, citra hasil keluaran dan beberapa informasi penting yang dibutuhkan untuk melakukan evaluasi.

4.2.4.1. Implementasi Antar Muka Proses Penyisipan *Watermark*

Implementasi antar muka proses penyisipan *watermark* bertujuan untuk memudahkan pengguna dalam menjalankan program. Antar muka pada proses penyisipan *watermark* terbagi menjadi tiga fungsi, yaitu untuk menampilkan citra masukan, menampilkan parameter masukan, dan menampilkan citra hasil proses penyisipan.

Tampilan hasil implementasi antar muka untuk proses penyisipan *watermark* ditunjukkan oleh Gambar 4.1. Hasil yang ditampilkan dari proses penyisipan *watermark* adalah citra *master share* dan citra *ownership share*. Pengguna dapat menyimpan citra

ownership share dengan nama file pada direktori tertentu. Sehingga dapat digunakan kembali pada program ekstraksi *watermark*. Di bawah panel kunci juga terdapat informasi total waktu *running* program penyisipan.

Gambar 4.1 Implementasi Antar Muka Proses Penyisipan *Watermark*

4.2.4.2. Implementasi Antar Muka Proses Ekstraksi *Watermark*

Implementasi antar muka proses ekstraksi bertujuan untuk memudahkan pengguna dalam menjalankan program ekstraksi. Antar muka pada proses ekstraksi *watermark* terbagi menjadi empat fungsi, yaitu untuk menampilkan citra masukan,

menampilkan parameter masukan, dan menampilkan citra hasil proses ekstraksi dan menampilkan hasil uji NC.

Tampilan hasil implementasi antar muka untuk proses ekstraksi *watermark* ditunjukkan oleh Gambar 4.2. Citra yang ditampilkan dari hasil proses ekstraksi *watermark* adalah citra *master share*, citra *stack*, dan citra *watermark* hasil ekstraksi yang sudah dikecilkan. Di bawah panel yang berisi parameter kunci juga terdapat informasi total waktu *running* program ekstraksi.

Nilai hasil uji NC dapat diketahui setelah pengguna melakukan uji NC terhadap citra *watermark* asli. Nilai hasil NC ditampilkan pada sebuah panel di bagian kanan.

Gambar 4.2 Implementasi Antar Muka Proses Ektraksi *Watermark*

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan hasil uji coba yang dilakukan pada aplikasi yang telah dikerjakan. Pembahasan pengujian meliputi lingkungan uji coba dan skenario uji coba. Skenario uji coba terdiri atas uji kebenaran dan uji kinerja serta analisis dari setiap pengujian.

5.1. Lingkungan Uji Coba

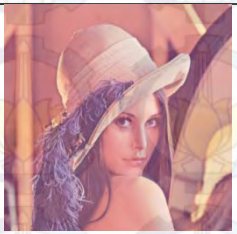
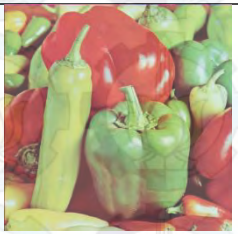
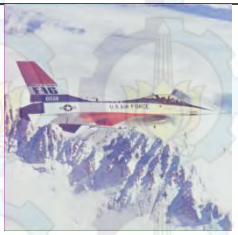
Subbab ini menjelaskan lingkungan yang digunakan untuk menguji kinerja terhadap aplikasi yang dibuat pada tugas akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang disebutkan sebagai berikut:

1. Perangkat keras
   a. Processor : Intel®Core™ i3 CPU M370 @2.40 GHz
   b. Memory (RAM) : 2.00 GB
   c. Tipe sistem : 32-bit Sistem Operasi
2. Perangkat lunak
   a. Sistem operasi : Microsoft Windows 8 Pro 32 bit
   b. Perangkat pengembang : MATLAB R2013a (8.1.0.604)

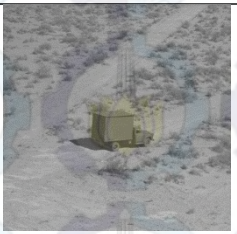
5.2. Data Uji Coba

Tahap uji coba sistem *blind watermarking* ini menggunakan 10 citra *grayscale* berukuran 512×512 yang diunduh dari database USC- SIPI. Beberapa citra *host* yang diambil dari database adalah citra berwarna, sehingga harus diubah menjadi citra *grayscale* terlebih dahulu. Sedangkan citra *watermark* yang digunakan berupa citra biner berukuran 64×64. Tabel 5.1 sampai Tabel 5.3 menunjukkan daftar citra yang dijadikan data uji coba.

Tabel 5.1 Data Uji Coba untuk Citra *Host* (Bagian 1)

No	Citra *Host*	No	Citra *Host*
1.	 lena.tiff	2.	 pepper.tiff
3.	 boat.512.tiff	4.	 baboon.tiff
5.	 bridge.tiff	6.	 plane.tiff
7.	 lake.tiff	8.	 house.tiff

Tabel 5.2 Data Uji Coba untuk Citra *Host* (Bagian 2)

No	Citra *Host*	No	Citra *Host*
9.	 tank.tiff	10.	 truck.tiff

Tabel 5.3 Data Uji Coba untuk Citra *Watermark*

No	Citra *Watermark*	No	Citra *Watermark*
1.		2.	

5.3. Skenario Uji Coba

Uji coba ini dilakukan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan seperti seharusnya. Uji coba didasarkan pada beberapa skenario untuk menguji kesesuaian dan kinerja aplikasi.

Secara garis besar, uji coba terdiri dari dua bagian, yaitu uji kebenaran dan uji kinerja.

5.3.1. Uji Kebenaran

Pada uji coba ini dilakukan pengujian terhadap kebenaran dan kesesuaian sistem yang telah dibuat. Uji coba kebenaran bertujuan untuk menunjukkan bahwa program dapat berjalan sebagaimana seharusnya. Citra *watermark* yang disisipkan dengan menggunakan kunci tertentu dan diekstraksi menggunakan kunci

yang bernilai sama akan menghasilkan citra *watermark* dengan citra *watermark* awal. Sementara itu jika proses ekstraksi dilakukan dengan menggunakan kunci yang berbeda akan menghasilkan citra *watermark* yang rusak.

Citra *host* yang dimasukkan berupa citra *grayscale* dengan format .tif, .jpg, .png atau .bmp. Sedangkan citra *watermark* yang dimasukkan berupa citra biner dengan format .bmp. Citra *host* yang diproses pada tahap penyisipan dengan menggunakan kunci yang telah ditentukan menghasilkan citra *ownership share*. Kunci yang digunakan dalam proses penyisipan berjumlah 5, yaitu A, B, α, β dan posisi VSS. Nilai A dan B digunakan sebagai kunci pada proses *cat map* dan memiliki kriteria tidak lebih dari panjang atau lebar citra *watermark*. Nilai α dan β digunakan sebagai orde pada transformasi 2D-DFRFT. Nilainya dapat diisi bilangan negatif, bilangan positif, bilangan bulat, bilangan desimal maupun bilangan pecahan. Sedangkan posisi VSS menentukan kombinasi posisi piksel pada blok-blok untuk pembentukan *master share*t (diagonal, horisontal dan vertikal).

Skenario uji kebenaran terdiri atas uji coba proses penyisipan dan proses ekstraksi. Dalam uji coba ini juga dilakukan analisis waktu komputasi untuk citra pada keadaan normal. Uji kebenaran juga dicoba pada proses ekstraksi dengan kunci yang salah. Masing-masing kunci akan dilihat pengaruhnya terhadap hasil ekstraksi yang didapatkan.

5.3.1.1. Uji Kebenaran Skenario 1: Uji Kebenaran Proses Penyisipan dan Proses Ekstraksi

Pada skenario 1 pengujian dilakukan dengan melakukan proses penyisipan kemudian dilanjutkan dengan proses ekstraksi. Kunci yang digunakan pada uji kebenaran ini adalah adalah sebagai berikut: A=32, B=12, $\alpha = 3/2$, $\beta = -1/2$ dan posisi VSS diagonal. Pada proses ekstraksi, citra *host* yang diklaim diekstraksi menggunakan citra *ownership share* dan lima kunci yang sama.

Tabel 5.10 menunjukkan hasil uji kebenaran untuk sebuah citra uji. Hasil uji untuk 9 citra lainnya ditampilkan pada Tabel A.1 sampai Tabel A.3 di bagian lampiran. Dari Tabel 5.10 dapat dilihat bahwa citra *watermark* yang dihasilkan pada proses ekstraksi sama persis denga citra *watermark* asli. Dari sini dapat dikatakan bahwa sistem sudah sesuai.

Tabel 5.4 Hasil Proses Penyisipan dan Ekstraksi pada Uji Kebenaran Sistem Menggunakan Kunci yang Benar

<table>
<tr><td colspan="2">Citra Host</td><td>Citra Watermark 1</td><td>Citra Watermark 2</td></tr>
<tr><td colspan="2">  </td><td>  </td><td>  </td></tr>
<tr><td>Stacked Image 1</td><td>Reduced Watermark 1</td><td>Stacked Image 2</td><td>Reduced Watermark 2</td></tr>
<tr><td>  </td><td> 
NC =1</td><td>  </td><td> 
NC =1</td></tr>
</table>

Tabel 5.5 menunjukkan waktu komputasi untuk program penyisipan dan program ekstraksi untuk semua citra pada keadaan normal. Kedua proses memiliki rata-rata waktu komputasi yang mirip, yaitu ±17.2 detik. Di bagian lampiran pada Tabel A.4 dan Tabel A.5 dapat dilihat detil waktu komputasi untuk masing-masing proses. Jika dilihat dari waktu komputasi yang dibutuhkan,

terbukti bahwa algoritma yang diusulkan tidak membutuhkan waktu yang lama untuk melakukan sekali proses penyisipan maupun proses ektraksi.

Tabel 5.5 Waktu Komputasi untuk Program Penyisipan dan Program Ekstraksi

Nama Citra	Waktu Penyisipan (s)	Waktu Ekstraksi (s)
Lena	17.238	17.283
Pepper	17.396	17.999
Boat	16.980	17.764
Baboon	16.461	17.237
Bridge	17.501	17.080
Plane	16.984	17.238
Lake	17.217	17.535
Tank	16.836	16.710
Truck	16.664	16.740
Rata-Rata	17.031	17.287

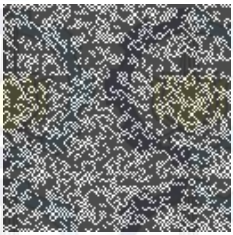
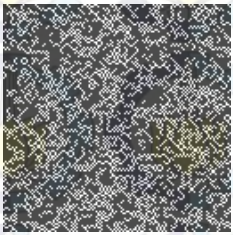
5.3.1.2. Uji Kebenaran Skenario 2: Uji Ekstraksi Menggunakan Citra *Ownership Share* yang Salah

Berkaitan dengan keunggulan algoritma yang diusulkan yang tidak mengubah apapun dari citra *host*, maka perlu dilakukan uji kebenaran terhadap kepemilikan citra *ownership share*. Uji kinerja ini bertujuan untuk membuktikan bahwa kepemilikan *ownership share* yang digunakan sebagai kunci algoritma *watermarking* hanya dapat digunakan pada citra pemilik yang sesungguhnya.

Dari Tabel 5.6 terlihat jelas bahwa hasil ekstraksi citra *watermark* tidak menunjukkan informasi apapun jika masukan citra *ownership share* yang digunakan salah. Ini membuktikan bahwa kepemilikan *ownership share* yang merupakan kunci

algoritma *watermarking* hanya dapat digunakan pada citra pemilik yang sesungguhnya.

Tabel 5.6 Hasil Uji Kebenaran Skenario 2: Uji Ekstraksi Menggunakan Citra *Ownership Share* yang Salah

Citra *Host*	*Stacked Image*	*Reduced watermark*
		
		
		

5.3.1.3. Uji Kebenaran Skenario 3: Uji Ekstraksi Menggunakan Kunci Transformasi 2D-DFRFT yang Salah

Uji kebenaran ini bertujuan untuk membuktikan bahwa fungsi transformasi 2D-DFRFT memiliki sifat yang sensitif terhadap hasil transformasi yang diperoleh. Sehingga sifat sensitif parameter 2D-DFRFT ini dapat digunakan sebagai kunci algoritma *watermarking*. Karena jika kunci yang digunakan salah, maka akan berpengaruh pada hasil ekstraksi yang salah juga.

Tabel 5.7 Hasil Uji Kebenaran Skenario 3: Uji Ekstraksi Menggunakan Kunci Transformasi 2D-DFRFT yang Salah

Stacked Image	*Reduced watermark*	Keterangan
		Alpha = 1 Beta = -1/2
		Alpha = 3/2 Beta = 1
		Alpha = 10 Beta = -15

Skenario ini menggunakan citra lena.tiff sebagai citra *host* asli. Parameter 2D-DFRFT yang digunakan pada proses penyisipan

yaitu α = 3/2 dan β = -1/2. Sedangkan pada proses ektraksi dilakukan uji kinerja menggunakan kombinasi parameter yang salah pada α saja, β saja dan kedua parameter α danβ. Hasil uji kebenaran skenario 2 dapat dilihat pada Tabel 5.7.

Pada Tabel 5.7 terlihat bahwa hasil ekstraksi citra watermark masih terlihat jelas meskipun parameter yang digunakan salah. Ini menunjukkan bahwa implementasi 2D-DFRFT kurang memberikan sensitifitas terhadap tingkat keamanan algoritma. Hal ini mungkin saja terjadi karena transformasi dilakukan pada subimage bukan pada citra secara keseluruhan. Selain itu fitur citra yang digunakan adalah hasil dekomposisi SVD. Nilai singular yang diperoleh melalui dekomposisi SVD tetap memberikan hasil yang tidak jauh berbeda meskipun sudah dilakukan transformasi dengan parameter yang berbeda.

5.3.1.4. Uji Kebenaran Skenario 4: Uji Ekstraksi Menggunakan Kunci *Cat Map* yang Salah

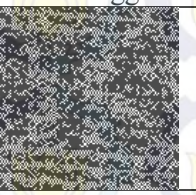
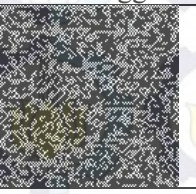
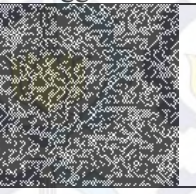
Uji kebenaran ini bertujuan untuk membuktikan bahwa fungsi pengacakan *cat map* memiliki sifat yang sensitif terhadap hasil ekstraksi yang diperoleh. Sehingga sifat sensitif kunci *cat map* ini dapat digunakan sebagai kunci algoritma *watermarking.* Karena jika kunci yang digunakan salah, maka hal ini akan berpengaruh pada hasil ekstraksi.

Skenario ini menggunakan citra lena.tiff sebagai citra host asli. Parameter kunci *cat map* yang digunakan pada proses penyisipan yaitu A = 32 dan B = 12. Sedangkan pada proses ektraksi dilakukan uji kinerja menggunakan kombinasi kunci yang salah pada kunci A saja, kunci B saja dan kedua kunci A dan B. Hasil uji kebenaran skenario 4 dapat dilihat pada Tabel 5.8.

Pada Tabel 5.8 terlihat bahwa hasil ekstraksi citra *watermark* tidak menunjukkan informasi apapun jika masukan kunci *cat map* yang digunakan salah. Ini membuktikan bahwa parameter kunci *cat map* memiliki sifat sensitif yang dapat digunakan untuk meningkatkan keamanan algoritma

watermarking. Sebuah citra hanya dapat diekstrak kembali hanya dengan memakai kunci yang sama pada proses penyisipan.

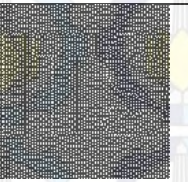
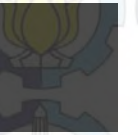
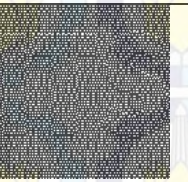
Tabel 5.8 Hasil Uji Kebenaran Skenario 3: Uji Ekstraksi Menggunakan Kunci *Cat Map* yang Salah

Hasil Ekstraksi Menggunakan kunci a=10		
		
Hasil Ekstraksi Menggunakan kunci b=5		
		
Hasil Ekstraksi Menggunakan kunci a=15dan b=5		
		

5.3.1.5. Uji Kebenaran Skenario 5: Uji Ekstraksi Menggunakan Posisi VSS yang Salah

Uji kinerja ini bertujuan untuk membuktikan bahwa posisi piksel yang digunakan pada VSS memiliki sifat yang sensitif terhadap hasil yang didapatkan. Artinya citra *watermark* hanya dapat diekstraksi jika posisi VSS yang digunakan pada proses ekstraksi sama dengan posisi VSS yang digunakan pada proses penyisipan. Sehingga sifat sensitif ini dapat digunakan sebagai kunci algoritma *watermarking*.

Tabel 5.9 Hasil Uji Kebenaran Skenario 4: Uji Ekstraksi Menggunakan Posisi VSS yang Salah

Posisi VSS	*Stacked Image*	*Reduced Watermark*
Horisontal		
Vertikal		
Diagonal		

Skenario ini menggunakan citra lena.tiff sebagai citra *host* asli. Posisi VSS yang digunakan pada proses penyisipan yaitu posisi diagonal (*type*=1). Sedangkan pada proses ektraksi dilakukan uji kinerja menggunakan posisi horisontal dan vertikal. Hasil uji kebenaran skenario 5 dapat dilihat pada Tabel 5.9.

Pada Tabel 5.9 terlihat jelas bahwa hasil ekstraksi citra *watermark* tidak menunjukkan informasi apapun jika masukan posisi VSS yang digunakan salah. Ini membuktikan bahwa citra *watermark* hanya dapat diekstraksi jika posisi VSS yang digunakan pada proses ekstraksi sama dengan posisi VSS yang digunakan pada proses penyisipan.

5.3.2. Uji Kinerja

Pada uji coba ini dilakukan pengujian kinerja program dari sisi kekokohan citra *watermark*. Citra *host* yang dimasukkan untuk uji coba berupa citra *grayscale* dengan format .tiff, kecuali untuk citra hasil kompresi dengan format .jpg. Citra *watermark* yang dimasukkan berupa citra biner nomor 1 yang tertera pada Tabel 5.3.

Skenario uji kinerja ketahanan citra *watermark* dilakukan dengan menerapkan berbagai kemungkinan gangguan yang menyerang citra *host*. Gangguan tersebut antar lain seperti penambahan derau, kompresi JPEG, *sharpening*, *blurring*, *resizing*, rotasi dan *cropping*.

Hasil uji coba diukur menggunakan besaran PSNR dan NC. Nilai besaran PSNR menunjukkan perbandingan kualitas antara citra *host* yang mengalami degradasi dengan citra *host* asli. Sedangkan nilai besaran NC menunjukkan tingkat kemiripan citra *watermark* hasil ekstraksi terhadap citra *watermark* asli. *Watermark* dinyatakan memilki ketahanan yangbaik jika logo yang tertera masih dapat dikenali dengan baik oleh mata manusia, atau jika diukur dengan nilai NC menunjukkan rentang nilai 0.65-1.

Detil setiap uji skenario dijelaskan lebih lanjut pada setiap subbab. Hasil uji coba yang ditampilkan pada masing-masing subbab berikut hanya berupa tabel ringkasan dan tabel untuk sebagian hasil ekstraksi. Tabel ringkasan berisi nilai rata-rata, nilai terbesar dan nilai terkecil dan tabel. Tabel hasil ekstraksi berisi tampilan citra *host* dan citra hasil ekstraksi. Detil keseluruhan uji coba dan lanjutan tabel hasil ekstraksi dapat dilihat pada Tabel A.6 sampai Tabel A.26 di bagian lampiran.

5.3.2.1. Uji Kinerja dan Evaluasi Skenario 1: Uji Ketahanan Citra *Watermark* terhadap *Blurring*

Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan *blurring* terhadap citra *host*. Proses *blurring* dilakukan dengan menggunakan *filter average* dengan ukuran *mask* yang berbeda-beda. Terdapat 5 macam *mask* yang digunakan, mulai ukuran 3×3 sampai 15×15. Hasil uji kinerja skenario 1 dapat dilihat pada Tabel 5.10 dan Tabel 5.11.

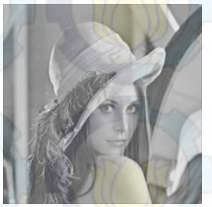
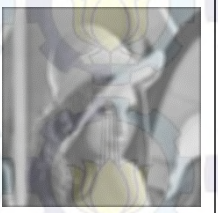
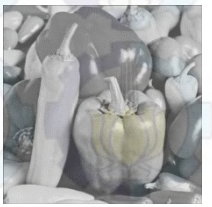
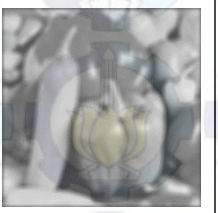
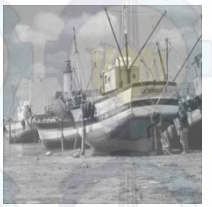
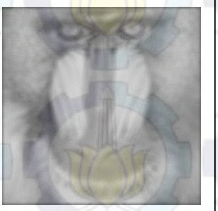
Tabel 5.10 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Gangguan *Blurring*

Ukuran *Mask*	**Average**		**Max**		**Min**	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
3×3	29.3345	0.9914	31.9362	0.9966	23.2596	0.9822
5×5	26.1295	0.9822	28.9921	0.9915	21.1234	0.9688
7×7	24.4674	0.9711	27.4441	0.9854	20.3796	0.9551
9×9	23.3838	0.958	26.398	0.9788	19.9578	0.9331
15×15	21.4455	0.9232	24.5115	0.9578	19.2116	0.8918

Dari Tabel 5.10 dapat diketahui bahwa semakin besar ukuran *mask* yang digunakan maka nilai PSNR citra juga akan semakin kecil dan berakibat terhadap nilai NC yang dihasilkan semakin turun. Tetapi meskipun citra sudah mengalami pengaburan dengan ukuran *mask* yang terbesar, nilai NC terendah yang dihasilkan masih menunjukkan angka yang tinggi, yaitu 0.8918.

Pada Tabel 5.11 juga dapat dilihat perbandingan citra *watermark* hasil ekstraksi untuk ukuran *mask* yang kecil dan terbesar. Citra *watermark* hasil ekstraksi terhadap gangguan *blurring* ternyata masih terlihat jelas. Ini membuktikan bahwa citra *watermark* yang disisipkan memiliki tingkat ketahanan yang baik terhadap gangguan *blurring* yang terjadi pada citra *host*.

Tabel 5.11 Hasil Uji Kinerja Skenario 1: Uji Ketahanan *Watermark* terhadap Gangguan *Blurring*

Blurring 3×3		*Blurring* 15×15	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9929		 0.9343
	 0.9966		 0.9578
	 0.9868		 0.9050
	 0.9822		 0.9063

5.3.2.2. Uji Kinerja dan Evaluasi Skenario 2: Uji Ketahanan Citra *Watermark* terhadap *Sharpening*

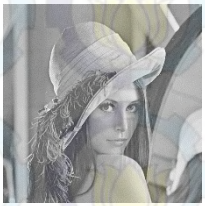
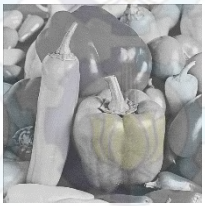
Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan *sharpening* terhadap citra *host*. Proses *sharpening* dilakukan dengan menggunakan fungsi *imsharpen* dengan parameter *strength* yang berbeda-beda. Parameter *strength* menentukan seberapa besar pengaruh *sharpening* yang diberikan. Masing-masing citra *host* diberi pengaruh *sharpening* sampai 5 tingkat. Hasil uji kinerja skenario 2 dapat dilihat pada Tabel 5.12 dan Tabel 5.13.

Tabel 5.12 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Gangguan *Sharpening*

Parameter *Sharpening*	***Average***		***Max***		***Min***	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
1	30.3355	0.9895	33.3686	0.9968	23.9036	0.9626
2	24.6788	0.9765	27.5067	0.99	18.5671	0.9229
3	21.6607	0.9665	24.3145	0.9856	16.0808	0.9038
4	19.6901	0.9552	22.2161	0.9788	14.6039	0.8865
5	18.2593	0.9452	20.6882	0.9751	13.597	0.8745

Dari Tabel 5.12 dapat diketahui bahwa semakin tinggi tingkat penajaman yang digunakan maka nilai PSNR citra juga akan semakin kecil dan berakibat terhadap nilai NC yang dihasilkan semakin turun. Tetapi meskipun citra sudah mengalami *sharpening* dengan tingkat yang tertinggi, nilai NC terendah yang dihasilkan masih baik, yaitu 0.8745. Pada Tabel 5.13 juga dapat dilihat perbandingan citra *watermark* hasil ekstraksi untuk tingkat *sharpening* terendah dan tertinggi. Citra *watermark* hasil ekstraksi terhadap gangguan *sharpening* ternyata masih terlihat jelas. Ini membuktikan bahwa citra *watermark* yang disisipkan memiliki tingkat ketahanan yang baik terhadap gangguan *sharpening* yang terjadi pada citra *host*.

Tabel 5.13 Hasil Uji Kinerja Skenario 2: Uji Ketahanan *Watermark* terhadap Gangguan *Sharpening*

Sharpening 1		*Sharpening* 5	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9924		 0.9441
	 0.9968		 0.9751
	 0.9866		 0.9263
	 0.9626		 0.8745

5.3.2.3. Uji Kinerja dan Evaluasi Skenario 3: Uji Ketahanan Citra *Watermark* terhadap Kompresi JPEG

Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan kompresi JPEG terhadap citra *host*. Kompresi dilakukan dengan menyimpan kembali file citra uji dalam format .jpg dengan kualitas yang berbeda-beda. Terdapat 5 macam kualitas kompresi yang digunakan, yaitu mulai 50% sampai 10%. Hasil uji kinerja skenario 3 dapat dilihat pada Tabel 5.14 dan Tabel 5.15.

Tabel 5.14 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Gangguan Kompresi JPEG

Kualitas Kompresi	***Average***		***Max***		***Min***	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
50%	33.2392	0.9957	35.9975	0.999	28.228	0.9934
40%	32.5478	0.9949	35.2168	0.999	27.3785	0.9917
30%	31.7105	0.9919	34.2818	0.9971	26.448	0.9839
20%	30.4522	0.9892	32.9611	0.9968	25.2638	0.9778
10%	28.1734	0.9803	30.4112	0.9944	23.4246	0.9526

Dari Tabel 5.14 dapat diketahui bahwa semakin kecil kualitas kompresi yang digunakan maka nilai PSNR citra juga akan semakin kecil dan berakibat terhadap nilai NC yang dihasilkan semakin turun. Tetapi meskipun citra sudah mengalami kompresi JPEG dengan kualitas 10%, nilai NC terendah yang dihasilkan masih menunjukkan angka yang tinggi, yaitu 0.9526.

Pada Tabel 5.15 juga dapat dilihat perbandingan citra *watermark* hasil ekstraksi untuk kualitas kompresi tertinggi dan terendah. Citra *watermark* hasil ekstraksi terhadap gangguan kompresi JPEG ternyata masih terlihat jelas. Ini membuktikan bahwa citra *watermark* yang disisipkan memiliki tingkat ketahanan yang baik terhadap gangguan kompresi JPEG yang terjadi pada citra *host*.

Tabel 5.15 Hasil Uji Kinerja Skenario 3: Uji Ketahanan *Watermark* terhadap Gangguan Kompresi JPEG

Quality 50%		*Quality* 10%	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9941		 0.9661
	 0.9980		 0.9895
	 0.9961		 0.9741
	 0.9941		 0.9661

5.3.2.4. Uji Kinerja dan Evaluasi Skenario 4: Uji Ketahanan Citra *Watermark* terhadap *Resizing*

Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan *resizing* terhadap citra *host*. Pada uji coba ini *resizing* dilakukan dengan menggunakan fungsi *imresize*. Masing-masing citra *host* diubah ukurannya mulai 256×256 sampai 32×32, kemudian citra *host* hasil *resizing* dikembalikan lagi pada ukuran 512×512. Hasil uji kinerja skenario 4 dapat dilihat pada Tabel 5.16 dan Tabel 5.17.

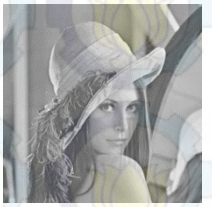
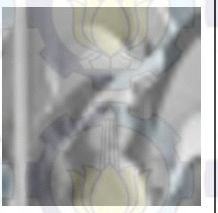
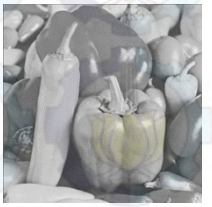
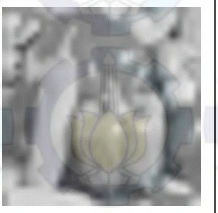
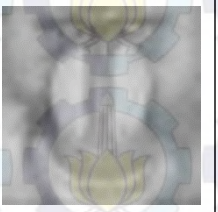
Tabel 5.16 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Gangguan *Resizing*

Resize to	***Average***		***Max***		***Min***	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
256×256	30.2131	0.9957	34.1092	0.9978	23.6253	0.9883
200×200	28.5726	0.993	32.0632	0.9968	22.438	0.9817
128×128	26.1634	0.9891	29.7542	0.9934	21.0822	0.9758
100×100	25.0672	0.982	28.9247	0.989	20.5813	0.9648
64×64	23.373	0.9721	27.6382	0.9807	19.9246	0.9531
32×32	21.2022	0.921	25.9946	0.9395	18.9784	0.8928

Dari Tabel 5.16 dapat diketahui bahwa semakin kecil ukuran citra hasil *resizing* maka nilai PSNR citra juga akan semakin kecil dan berakibat terhadap nilai NC yang dihasilkan semakin turun. Tetapi meskipun citra sudah mengalami resizing sampai ukuran 32×32 nilai NC terendah yang dihasilkan masih menunjukkan angka yang tinggi , yaitu 0.8928.

Pada Tabel 5.17 juga dapat dilihat perbandingan citra *watermark* hasil ekstraksi untuk ukuran *resizing* yang terbesar dan terkecil. Citra *watermark* hasil ekstraksi terhadap gangguan *resizing* ternyata masih terlihat jelas. Ini membuktikan bahwa citra *watermark* yang disisipkan memiliki tingkat ketahanan yang baik terhadap gangguan *resizing*.

Tabel 5.17 Hasil Uji Kinerja Skenario 4: Uji Ketahanan *Watermark* terhadap Gangguan *Resizing*

Resize to 256×256		*Resize to* 32×32	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9976		 0.9343
	 0.9978		 0.9395
	 0.9956		 0.9097
	 0.9883		 0.8928

5.3.2.5. Uji Kinerja dan Evaluasi Skenario 5: Uji Ketahanan Citra *Watermark* terhadap Penambahan Derau

Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan penambahan derau terhadap citra *host*. Pada uji coba ini masing-masing citra *host* diberi derau jenis *gaussian* dan *salt & pepper*. Derau jenis *gaussian* diimplementasikan dengan *mean*=0 dan *variance*=0.03. Penambahan derau ini mengakibatkan citra *host* baru yang dihasilkan mengalami penurunan kualitas dengan nilai rata-rata PSNR ±15 dB. Sedangkan derau jenis *salt & pepper* diimplementasikan dengan *density* 0.03. Penambahan derau ini mengakibatkan citra *host* baru yang dihasilkan mengalami penurunan kualitas dengan nilai rata-rata PSNR ±18 dB. Hasil uji kinerja skenario 5 dapat dilihat pada Tabel 5.18 dan Tabel 5.19.

Tabel 5.18 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Penambahan Derau

Jenis Derau	Average		Max		Min	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
Gaussian	15.7516	0.9700	15.991	0.9863	15.4786	0.9434
Salt & Pepper	18.3914	0.9711	18.8175	0.9905	17.8591	0.9507

Dari Tabel 5.18 dapat diketahui bahwa nilai NC terendah yang dihasilkan oleh citra yang diberi derau jenis *gaussian* dan *salt & pepper* masih menunjukkan angka yang tinggi, yaitu 0.9434 dan 0.9507. Dari Tabel 5.19 juga dapat dilihat bahwa citra *watermark* hasil ekstraksi terhadap penambahan dua jenis derau tersebut juga masih masih terlihat jelas. Ini membuktikan bahwa citra *watermark* yang disisipkan memiliki tingkat ketahanan yang sangat baik terhadap penambahan derau.

Tabel 5.19 Hasil Uji Kinerja Skenario 5: Uji Ketahanan *Watermark* terhadap Penambahan Derau

Gaussian Noise		*Salt & Pepper*	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9575		 0.9653
	 0.9802		 0.9844
	 0.9509		 0.9690
	 0.9434		 0.9561

5.3.2.6. Uji Kinerja dan Evaluasi Skenario 6: Uji Ketahanan Citra *Watermark* terhadap Rotasi

Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan rotasi terhadap citra *host*. Pada uji coba ini, masing-masing citra *host* mengalami 6 kali rotasi dengan sudut yang berbeda-beda. Fungsi rotasi dilakukan mulai sudut 3º sampai 13º searah jarum jam. Gangguan rotasi ini mengakibatkan beberapa bagian dari citra *host* hilang karena mengalami *cropping* untuk mempertahankan ukuran citra *host*. Hasil uji kinerja skenario 6 dapat dilihat pada Tabel 5.20 dan Tabel 5.21.

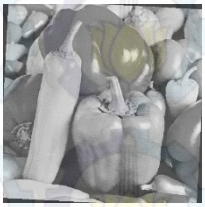
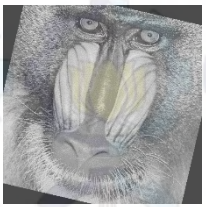
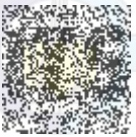
Tabel 5.20 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Rotasi

Sudut Rotasi	Average		Max		Min	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
Rotasi 3º	15.8107	0.8453	19.0935	0.8682	13.7884	0.8267
Rotasi 5º	14.3271	0.7923	17.7917	0.8152	12.3378	0.7673
Rotasi 7º	13.4085	0.7531	16.903	0.7947	11.5586	0.707
Rotasi 9º	12.7551	0.7186	16.3013	0.7678	11.0416	0.6533
Rotasi 11º	12.261	0.694	15.7894	0.7544	10.5019	0.6204
Rotasi 13º	11.8735	0.6722	15.3404	0.7322	10.0807	0.5906

Dari Tabel 5.20 dapat diketahui bahwa semakin besar sudut rotasi yang digunakan maka nilai PSNR citra juga akan semakin kecil dan berakibat terhadap nilai NC yang dihasilkan semakin turun. Nilai NC terkecil yang dihasilkan oleh citra dengan gangguan rotasi 3º masih menunjukkan angka yang cukup tinggi, yaitu 0.8267. Sedangkan nilai terkecil yang dihasilkan oleh citra dengan gangguan rotasi 13º sudah menunjukkan angka yang rendah, yaitu 0.5906. Namun jika dilihat dari nilai-rata-rata NC yang dihasilkan masih cukup baik, yaitu 0.6722. Pada Tabel 5.21 juga dapat dilihat bahwa citra *watermark* hasil ekstraksi untuk gangguan rotasi 3º masih terlihat jelas. Sedangkan beberapa citra *watermark* hasil ekstraksi untuk gangguan rotasi 13º terlihat

kurang jelas. Dari sini terbukti bahwa citra *watermark* memiliki ketahanan yang baik terhadap gangguan rotasi sampai 13º.

Tabel 5.21 Hasil Uji Kinerja Skenario 6: Uji Ketahanan *Watermark* terhadap Gangguan Rotasi

Rotasi 3º		Rotasi 13º	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.8682		 0.6475
	 0.8533		 0.6326
	 0.8391		 0.7209
	 0.8318		 0.6592

5.3.2.7. Uji Kinerja dan Evaluasi Skenario 7: Uji Ketahanan Citra *Watermark* terhadap *Cropping*

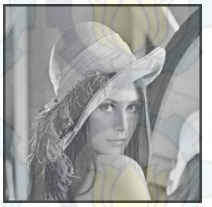
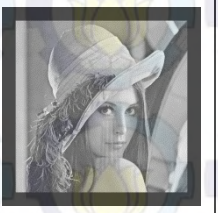
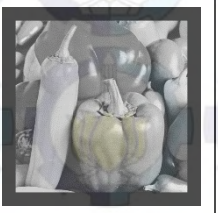
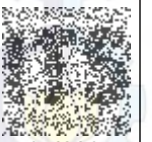
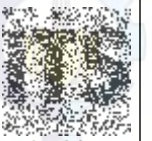
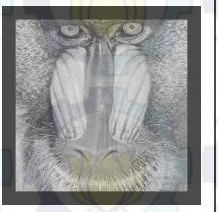
Uji kinerja ini bertujuan untuk menguji kekokohan *watermark* setelah dilakukan *cropping* terhadap citra *host*. Pada uji coba ini masing-masing citra *host* mengalami 5 macam fungsi *cropping* dengan persentase yang berbeda-beda. Fungsi *cropping* dilakukan pada bagian tepi citra *host* dengan luas mulai 5% sampai 25% dari luas citra *host*. Sehingga semakin besar persentase *cropping* semakin banyak pula bagian dari citra *host* yang hilang. Hasil uji kinerja skenario 7 dapat dilihat pada Tabel 5.22 dan Tabel 5.23.

Tabel 5.22 Ringkasan Hasil Uji Coba Ketahanan *Watermark* Terhadap Gangguan *Cropping*

Presentase *Cropping*	**Average**		**Max**		**Min**	
	PSNR (dB)	NC	PSNR (dB)	NC	PSNR (dB)	NC
5%	18.7456	0.9337	20.7291	0.9658	16.0398	0.8965
10%	15.3408	0.8543	17.3627	0.927	12.5917	0.7834
15%	13.4758	0.7951	15.5489	0.9048	10.736	0.7139
20%	12.2052	0.7524	14.261	0.864	9.4327	0.6494
25%	11.1223	0.6957	13.1574	0.8274	8.3526	0.5833

Dari Tabel 5.22 dapat diketahui bahwa semakin besar persentase *cropping* maka nilai PSNR citra juga akan semakin kecil dan berakibat terhadap nilai NC yang dihasilkan semakin turun. Nilai NC terkecil yang dihasilkan oleh citra dengan gangguan *cropping* 5% masih menunjukkan angka yang tinggi, yaitu 0.8965. Sedangkan nilai terkecil yang dihasilkan oleh citra dengan gangguan *cropping* 25% sudah menunjukkan angka yang rendah, yaitu 0.5833. Namun jika dilihat dari nilai-rata-rata NC yang dihasilkan masih cukup baik, yaitu 0.6957.

Tabel 5.23 Hasil Uji Kinerja Skenario 7: Uji Ketahanan *Watermark* terhadap Gangguan *Cropping*

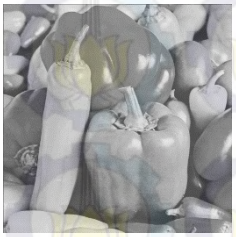
Cropping 5%		*Cropping* 25%	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9373		 0.7390
	 0.9448		 0.6990
	 0.9304		 0.7285
	 0.9280		 0.6492

Pada Tabel 5.23 juga dapat dilihat bahwa citra *watermark* hasil ekstraksi untuk gangguan *cropping* 5% masih terlihat jelas. Sedangkan beberapa citra *watermark* hasil ekstraksi untuk gangguan *cropping* 25% terlihat kurang jelas. Dari sini terbukti bahwa citra *watermark* yang disisipkan memiliki ketahanan yang baik terhadap gangguan *cropping* sampai persentase 25%.

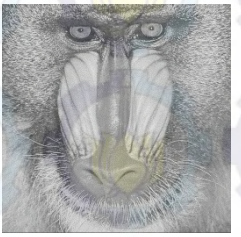
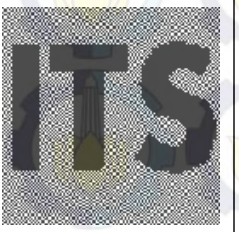
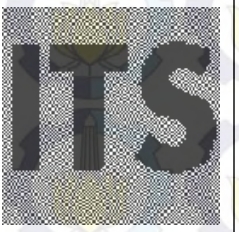
Halaman ini sengaja dikosongkan

LAMPIRAN

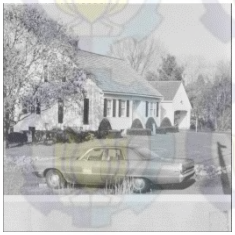
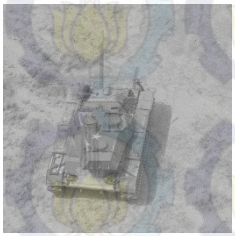
Tabel A.1 Hasil Ekstraksi Citra *Watermark* Menggunakan Kunci yang Benar (Bagian 1)

Citra *Host*	Citra *Water-mark*	*Stacked Image*	*Reduced Water-mark*
			
			
			

Tabel A.2 Hasil Ekstraksi Citra *Watermark* Menggunakan Kunci yang Benar (Bagian 2)

Citra *Host*	Citra *Water-mark*	*Stacked Image*	*Reduced Water-mark*
			
			
			

Tabel A.3 Hasil Ekstraksi Citra *Watermark* Menggunakan Kunci yang Benar (Bagian 3)

Citra *Host*	Citra *Water-mark*	*Stacked Image*	*Reduced Water-mark*
			
			
			

Tabel A.4 Waktu *Running* Program Penyisipan Citra *Watermark*

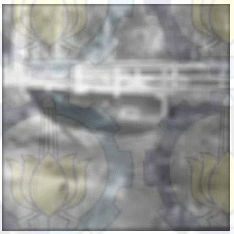
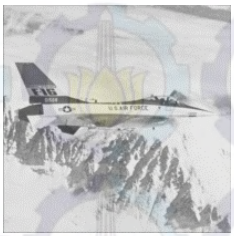
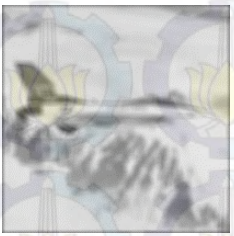
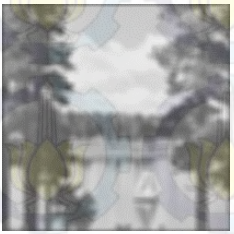
Nama Citra	t1*(s)	t2*(s)	Waktu Penyisipan(s)
Lena	17.235	0.004	17.238
Pepper	17.396	0.001	17.396
Boat	16.980	0.001	16.980
Baboon	16.459	0.001	16.461
Bridge	17.500	0.001	17.501
Plane	16.984	0.001	16.985
Lake	17.215	0.001	17.217
House	17.635	0.001	17.636
Tank	16.835	0.001	16.836
Truck	16.663	0.001	16.664

Tabel A.5 Waktu *Running* Program Ekstraksi Citra *Watermark*

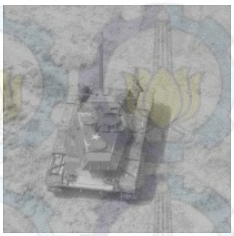
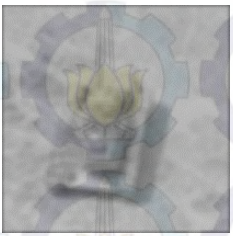
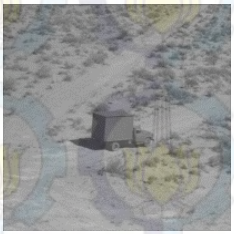
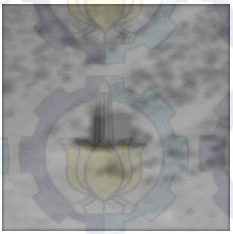
Nama Citra	t1**(s)	t2**(s)	Waktu Ekstraksi(s)
Lena	17.238	0.045	17.283
Pepper	17.950	0.050	17.999
Boat	17.717	0.046	17.764
Baboon	17.181	0.055	17.237
Bridge	17.031	0.049	17.080
Plane	17.192	0.046	17.238
Lake	17.486	0.049	17.535
House	17.566	0.047	17.613
Tank	16.662	0.048	16.710
Truck	16.694	0.046	16.740

Keterangan
t1* : waktu *running* pembentukan *master share* pada proses penyisipan
t1** : waktu *running* pembentukan *master share* pada proses ekstraksi
t2* : waktu *running* program pembentukan *ownership share*
t2** : waktu *running* program ekstraksi *watermark*

Tabel A.6 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Blurring* (Bagian 1)

Blurring 3×3		*Blurring* 15×15	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9915		 0.9297
	 0.9937		 0.9124
	 0.9961		 0.9553

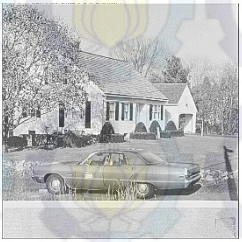
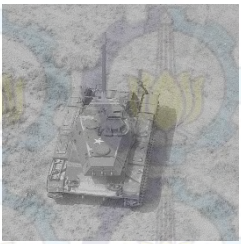
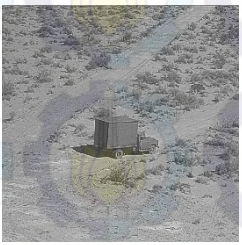
Tabel A.7 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Blurring* (Bagian 2)

Blurring 3×3		*Blurring* 15×15	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9934		 0.9141
	 0.9883		 0.8918
	 0.9922		 0.9248

Tabel A.8 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Sharpening* (Bagian 1)

Sharpening 1		*Sharpening* 5	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9883		 0.9280
	 0.9939		 0.9736
	 0.9934		 0.9656

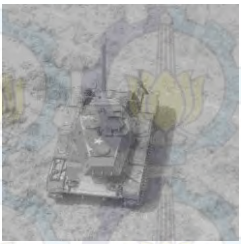
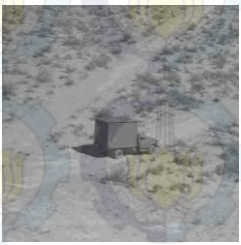
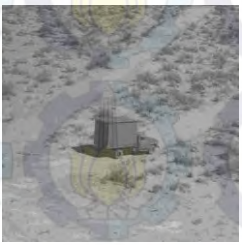
Tabel A.9 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Sharpening* (Bagian 2)

Sharpening 1		*Sharpening* 5	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9922		 0.9600
	 0.9934		 0.9534
	 0.9958		 0.9509

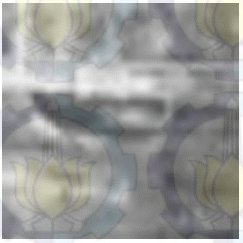
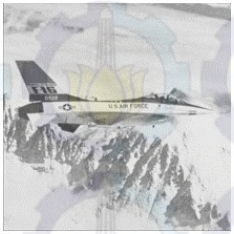
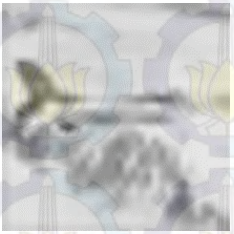
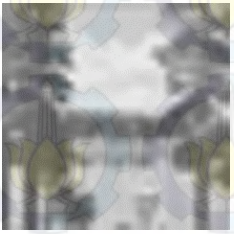
Tabel A.10 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan Kompresi JPEG (Bagian 1)

Quality 50%		*Quality* 10%	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9963		 0.9885
	 0.9968		 0.9875
	 0.9990		 0.9924

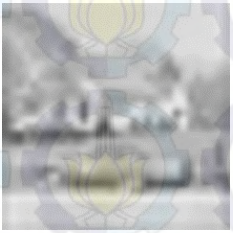
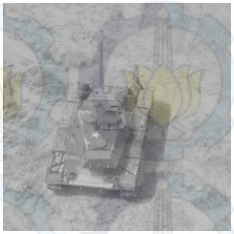
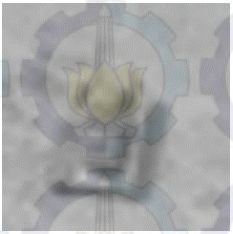
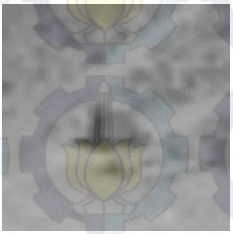
Tabel A.11 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan Kompresi JPEG (Bagian 2)

Quality 50%		*Quality* 10%	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9961		 0.9944
	 0.9934		 0.9922
	 0.9934		 0.9526

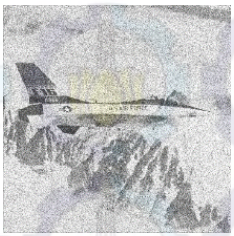
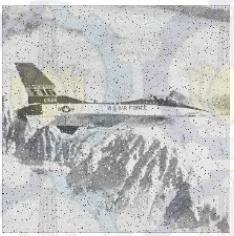
Tabel A.12 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Rezising* (Bagian 1)

Resize to 256×256		*Resize to* 32×32	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9956		 0.9138
	 0.9971		 0.9221
	 0.9978		 0.9380

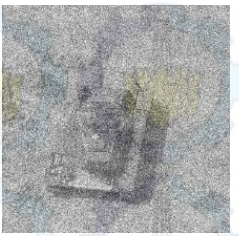
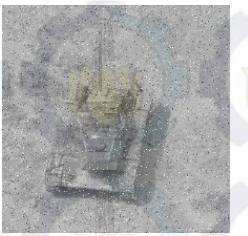
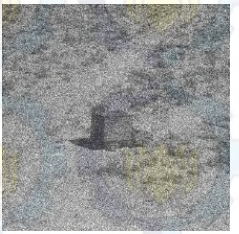
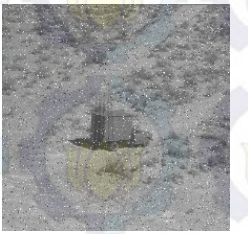
Tabel A.13 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Resizing* (Bagian 2)

Resize to 256×256		*Resize to* 32×32	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9961		 0.9202
	 0.9956		 0.9299
	 0.9954		 0.9094

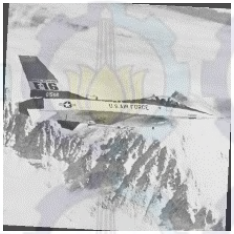
Tabel A.14 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Noise Addition* (Bagian 1)

Gaussian Noise		*Salt & Pepper*	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9666		 0.9746
	 0.9805		 0.9895
	 0.9863		 0.9905

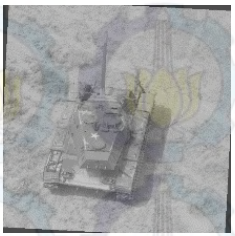
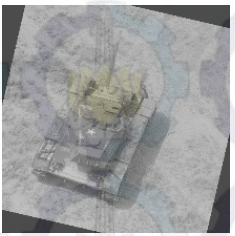
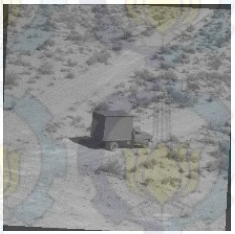
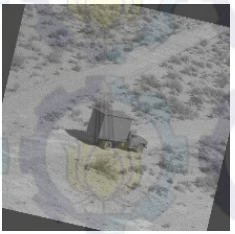
Tabel A.15 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Noise Addition* (Bagian 2)

Gaussian Noise		*Salt & Pepper*	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9675		 0.9775
	 0.9234		 0.9507
	 0.9382		 0.9531

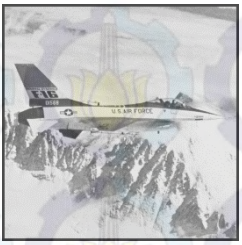
Tabel A.16 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan Rotasi (Bagian 1)

Rotasi 3º		Rotasi 13º	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.8325		 0.6987
	 0.8555		 0.6616
	 0.8425		 0.7112

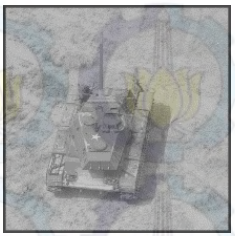
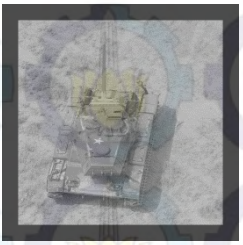
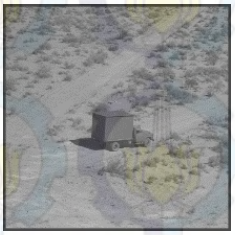
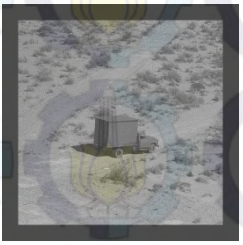
Tabel A.17 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan Rotasi (Bagian 2)

Rotasi 3º		Rotasi 13º	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.8411		 0.5906
	 0.8618		 0.7322
	 0.8267		 0.6677

Tabel A.18 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Cropping* (Bagian 1)

Cropping 5%		*Cropping* 25%	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9536		 0.7410
	 0.9348		 0.7056
	 0.9658		 0.8274

Tabel A.19 Hasil Ekstraksi Citra *Watermark* Terhadap Gangguan *Cropping* (Bagian 2)

Cropping 5%		*Cropping* 25%	
Citra *Host*	Hasil Ekstraksi	Citra *Host*	Hasil Ekstraksi
	 0.9175		 0.5833
	 0.8965		 0.5977
	 0.9285		 0.6858

Tabel A.20 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Gangguan *Blurring*

	***Blurring* 3×3**		***Blurring* 5×5**		***Blurring* 7×7**		***Blurring* 9×9**		***Blurring* 15×15**	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**
Lena	31.9362	0.9929	28.2908	0.9834	26.3196	0.9756	25.0248	0.9668	22.6451	0.9343
Pepper	31.5244	0.9966	28.2897	0.9915	26.3453	0.9854	24.9428	0.9775	22.2446	0.9578
Boat	28.8577	0.9868	25.5426	0.9719	23.8365	0.9556	22.7920	0.9409	21.0624	0.9050
Baboon	23.2596	0.9822	21.1234	0.9688	20.3796	0.9551	19.9578	0.9431	19.2116	0.9063
Bridge	25.8379	0.9915	23.1786	0.9824	21.8832	0.9712	21.0582	0.9614	19.5634	0.9297
Plane	30.8302	0.9937	26.5041	0.9863	24.2753	0.9695	22.8509	0.9500	20.4360	0.9124
Lake	29.3564	0.9961	25.6693	0.9902	23.6540	0.9846	22.3419	0.9788	20.0867	0.9553
House	29.1788	0.9934	25.3939	0.9871	23.5425	0.9800	22.3771	0.9629	20.3272	0.9141
Tank	30.7499	0.9883	28.3107	0.9761	26.9937	0.9587	26.0949	0.9331	24.3662	0.8918
Truck	31.8143	0.9922	28.9921	0.9839	27.4441	0.9751	26.3980	0.9656	24.5115	0.9248
Min	23.2596	0.9822	21.1234	0.9688	20.3796	0.9551	19.9578	0.9331	19.2116	0.8918
Max	31.9362	0.9966	28.9921	0.9915	27.4441	0.9854	26.398	0.9788	24.5115	0.9578
Average	29.3345	0.9914	26.1295	0.9822	24.4674	0.9711	23.3838	0.958	21.4455	0.9232

Tabel A.21 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Gangguan *Sharpening*

	***Sharpening* 1**		***Sharpening* 2**		***Sharpening* 3**		***Sharpening* 4**		***Sharpening* 5**	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**
Lena	33.3686	0.9924	27.5067	0.9802	24.3145	0.9692	22.2161	0.9563	20.6882	0.9441
Pepper	32.5849	0.9968	27.1506	0.9900	24.1367	0.9856	22.0921	0.9788	20.5623	0.9751
Boat	30.0585	0.9866	24.5503	0.9700	21.5584	0.9556	19.5845	0.9385	18.1351	0.9263
Baboon	23.9036	0.9626	18.5671	0.9229	16.0808	0.9038	14.6039	0.8865	13.5970	0.8745
Bridge	26.6482	0.9883	21.0750	0.9734	18.1319	0.9595	16.2628	0.9434	14.9563	0.9280
Plane	31.5319	0.9939	25.8866	0.9863	23.0545	0.9810	21.2855	0.9768	19.9917	0.9736
Lake	29.6076	0.9934	23.8291	0.9829	20.7760	0.9761	18.8426	0.9705	17.4845	0.9656
House	29.5678	0.9922	23.9783	0.9836	21.0425	0.9758	19.1340	0.9673	17.7866	0.9600
Tank	32.8494	0.9934	26.9055	0.9861	23.5012	0.9792	21.1470	0.9685	19.3727	0.9534
Truck	33.2340	0.9958	27.3387	0.9897	24.0107	0.9788	21.7323	0.9658	20.0181	0.9509
Min	23.9036	0.9626	18.5671	0.9229	16.0808	0.9038	14.6039	0.8865	13.597	0.8745
Max	33.3686	0.9968	27.5067	0.99	24.3145	0.9856	22.2161	0.9788	20.6882	0.9751
Average	30.3355	0.9895	24.6788	0.9765	21.6607	0.9665	19.6901	0.9552	18.2593	0.9452

Tabel A.22 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Gangguan Kompresi JPEG

	***Quality* 50%**		***Quality* 40%**		***Quality* 30%**		***Quality* 20%**		***Quality* 10%**	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**
Lena	35.8082	0.9941	35.1283	0.9939	34.2818	0.9863	32.9611	0.9846	30.4112	0.9661
Pepper	34.7706	0.9980	34.2135	0.9990	33.5436	0.9963	32.4357	0.9954	30.1396	0.9895
Boat	33.4953	0.9961	32.7532	0.9924	31.8313	0.9897	30.4935	0.9934	28.1346	0.9741
Baboon	28.2280	0.9941	27.3785	0.9939	26.4480	0.9863	25.2638	0.9846	23.4246	0.9661
Bridge	29.5437	0.9963	28.8533	0.9973	28.0763	0.9954	27.0131	0.9951	25.1270	0.9885
Plane	35.9975	0.9968	35.2168	0.9954	34.1906	0.9971	32.5732	0.9949	29.7646	0.9875
Lake	32.5316	0.9990	31.9615	0.9978	31.2241	0.9968	30.0706	0.9968	27.8220	0.9924
House	34.0258	0.9961	33.1803	0.9954	32.2070	0.9951	30.6686	0.9883	28.0275	0.9944
Tank	33.5899	0.9934	33.0269	0.9919	32.3324	0.9922	31.2946	0.9778	29.3826	0.9922
Truck	34.4014	0.9934	33.7658	0.9917	32.9694	0.9839	31.7474	0.9810	29.5001	0.9526
Min	28.228	0.9934	27.3785	0.9917	26.448	0.9839	25.2638	0.9778	23.4246	0.9526
Max	35.9975	0.999	35.2168	0.999	34.2818	0.9971	32.9611	0.9968	30.4112	0.9944
Average	33.2392	0.9957	32.5478	0.9949	31.7105	0.9919	30.4522	0.9892	28.1734	0.9803

Tabel A.23 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Gangguan *Resizing*

	***Resize to* 256×256**		***Resize to* 200×200**		***Resize to* 128×128**		***Resize to* 100×100**		***Resize to* 64×64**		***Resize to* 32×32**	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**
Lena	34.1092	0.9976	32.0632	0.9963	28.8434	0.9934	27.4293	0.9890	25.1593	0.9785	21.9907	0.9343
Pepper	31.7423	0.9978	30.2101	0.9968	27.9258	0.9932	26.6920	0.9890	24.5685	0.9795	21.2168	0.9395
Boat	29.9365	0.9956	28.1680	0.9907	25.5331	0.9849	24.4395	0.9780	22.8256	0.9668	20.8140	0.9097
Baboon	23.6253	0.9883	22.4380	0.9817	21.0822	0.9758	20.5813	0.9648	19.9246	0.9531	19.1353	0.8928
Bridge	26.4965	0.9956	25.0728	0.9929	23.0640	0.9900	22.1622	0.9792	20.8543	0.9705	18.9784	0.9138
Plane	31.2548	0.9971	29.1891	0.9956	26.1976	0.9932	24.7561	0.9878	22.6398	0.9797	20.2693	0.9221
Lake	29.6532	0.9978	27.8920	0.9966	25.1145	0.9924	23.7534	0.9888	21.6809	0.9807	19.0955	0.9380
House	29.0730	0.9961	27.2156	0.9932	24.5933	0.9900	23.4809	0.9805	21.7372	0.9700	19.8821	0.9202
Tank	32.8518	0.9956	31.6036	0.9932	29.7542	0.9905	28.9247	0.9817	27.6382	0.9712	25.9946	0.9299
Truck	33.3885	0.9954	31.8736	0.9934	29.5261	0.9875	28.4528	0.9810	26.7013	0.9707	24.6450	0.9094
Min	23.6253	0.9883	22.438	0.9817	21.0822	0.9758	20.5813	0.9648	19.9246	0.9531	18.9784	0.8928
Max	34.1092	0.9978	32.0632	0.9968	29.7542	0.9934	28.9247	0.989	27.6382	0.9807	25.9946	0.9395
Average	30.2131	0.9957	28.5726	0.993	26.1634	0.9891	25.0672	0.982	23.373	0.9721	21.2022	0.921

Tabel A.24 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Penambahan Derau

	Gaussian		***Salt & Pepper***	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**
Lena	15.5545	0.9575	18.4683	0.9653
Pepper	15.641	0.9802	18.3518	0.9844
Boat	15.5740	0.9509	18.4865	0.9690
Baboon	15.4786	0.9434	18.5578	0.9561
Bridge	15.6961	0.9666	18.3014	0.9746
Plane	15.9910	0.9805	17.8591	0.9895
Lake	15.9117	0.9863	17.9652	0.9905
House	15.7661	0.9675	18.3162	0.9775
Tank	15.9910	0.9805	18.8175	0.9507
Truck	15.9117	0.9863	18.7898	0.9531
Min	15.4786	0.9434	17.8591	0.9507
Max	15.991	0.9863	18.8175	0.9905
Average	15.7516	0.9700	18.3914	0.9711

Tabel A.25 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Gangguan Rotasi

	Rotasi 3º		**Rotasi 5º**		**Rotasi 7º**		**Rotasi 9º**		**Rotasi 11º**		**Rotasi 13º**	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**
Lena	16.1425	0.8682	14.4103	0.8057	13.3295	0.7512	12.5574	0.7029	12.0402	0.6692	11.6830	0.6475
Pepper	15.2863	0.8533	13.3338	0.7852	12.2302	0.7388	11.5034	0.6904	10.9730	0.6560	10.5473	0.6326
Boat	15.6724	0.8391	14.2129	0.8064	13.3129	0.7781	12.7488	0.7539	12.3439	0.7371	12.0220	0.7209
Baboon	15.2366	0.8318	14.4089	0.7834	13.8408	0.7500	13.3148	0.7161	12.8618	0.6824	12.4968	0.6592
Bridge	14.6115	0.8325	13.5292	0.7952	12.9316	0.7690	12.3749	0.7405	11.8841	0.7185	11.5059	0.6987
Plane	14.5193	0.8555	12.8557	0.8035	11.7718	0.7537	11.0416	0.7119	10.5019	0.6863	10.0807	0.6616
Lake	13.7884	0.8425	12.3378	0.7864	11.5586	0.7524	11.0890	0.7356	10.7932	0.7197	10.5919	0.7112
House	15.0200	0.8411	13.2925	0.7673	12.2098	0.7070	11.4307	0.6533	10.8547	0.6204	10.4076	0.5906
Tank	18.7366	0.8618	17.0978	0.8152	15.9968	0.7947	15.1893	0.7678	14.5673	0.7544	14.0597	0.7322
Truck	19.0935	0.8267	17.7917	0.7742	16.9030	0.7356	16.3013	0.7136	15.7894	0.6956	15.3404	0.6677
Min	13.7884	0.8267	12.3378	0.7673	11.5586	0.707	11.0416	0.6533	10.5019	0.6204	10.0807	0.5906
Max	19.0935	0.8682	17.7917	0.8152	16.903	0.7947	16.3013	0.7678	15.7894	0.7544	15.3404	0.7322
Average	15.8107	0.8453	14.3271	0.7923	13.4085	0.7531	12.7551	0.7186	12.261	0.694	11.8735	0.6722

Tabel A.26 Hasil Uji Kinerja Ketahanan *Watermark* terhadap Gangguan *Cropping*

	***Cropping* 5%**		***Cropping* 10%**		***Cropping* 15%**		***Cropping* 20%**		***Cropping* 25%**	
	PSNR (dB)	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**	**PSNR (dB)**	**NC**
Lena	19.3423	0.9373	16.0281	0.8774	14.1685	0.8289	12.9293	0.7905	11.7853	0.7390
Pepper	18.5376	0.9448	15.1252	0.8960	13.2639	0.8411	12.1344	0.7690	11.2200	0.6990
Boat	18.8590	0.9304	15.4976	0.8557	13.6804	0.7974	12.3888	0.7761	11.2865	0.7285
Baboon	19.1797	0.9280	15.7110	0.8181	13.8091	0.7307	12.5323	0.6902	11.5082	0.6492
Bridge	19.9386	0.9536	16.5437	0.8936	14.7188	0.8440	13.4593	0.8018	12.2915	0.7410
Plane	16.0398	0.9348	12.5917	0.8552	10.7360	0.7910	9.4327	0.7695	8.3526	0.7056
Lake	19.7719	0.9658	16.3277	0.9270	14.2568	0.9048	12.8303	0.8640	11.6827	0.8274
House	16.8416	0.9175	13.2965	0.7834	11.4529	0.7185	10.2004	0.6494	9.1108	0.5833
Tank	18.2160	0.8965	14.9233	0.7939	13.1222	0.7139	11.8834	0.6750	10.8278	0.5977
Truck	20.7291	0.9285	17.3627	0.8425	15.5489	0.7803	14.2610	0.7385	13.1574	0.6858
Min	16.0398	0.8965	12.5917	0.7834	10.736	0.7139	9.4327	0.6494	8.3526	0.5833
Max	20.7291	0.9658	17.3627	0.927	15.5489	0.9048	14.261	0.864	13.1574	0.8274
Average	18.7456	0.9337	15.3408	0.8543	13.4758	0.7951	12.2052	0.7524	11.1223	0.6957

Halaman ini sengaja dikosongkan

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan perangkat lunak dan hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain itu juga disampaikan saran yang untuk pengembangan perangkat lunak lebih lanjut.

6.1. Kesimpulan

Kesimpulan yang diperoleh berdasarkan hasil uji coba dan evaluasi adalah sebagai berikut:

1. Teknik *visual cryptography* yang diusulkan dapat melakukan penyisipan citra *watermark* tanpa harus mengubah citra *host*, sehingga kualitas citra *host* tetap tinggi.
2. Proses ekstraksi citra *watermark* juga dapat dilakukan secara *blind* dengan cara yang lebih sederhana, sehingga waktu komputasinya cepat.
3. Pada keadaan normal (tanpa perlakukan khusus) hasil ekstraksi citra *watermark* adalah sama persis dengan citra *watermark* asli sebelum disisipkan.
4. Proses ekstraksi citra *watermark* hanya dapat dilakukan dengan baik jika menggunakan citra *ownership* yang sah dan kunci yang benar.
5. Citra *watermark* yang disisipkan memiliki ketahanan yang sangat baik terhadap gangguan seperti *blurring*, *sharpening*, kompresi JPEG, *resizing* dan penambahan derau, yaitu dengan nilai NC terendah untuk masing-masing gangguan yang berada di kisaran 0.9.
6. Citra *watermark* yang disisipkan memiliki ketahanan yang baik terhadap gangguan rotasi sampai 13^{o}.
7. Citra *watermark* yang disisipkan memiliki ketahanan yang baik terhadap gangguan *cropping* sampai 25% dari luas citra asli.

6.2. Saran

Beberapa saran yang hendak disampaikan untuk pengembangan perangkat lunak lebih lanjut adalah sebagai berikut:

1. Perlu dikembangakan metode yang dapat meningkatkan ketahanan citra *watermark* dari serangan geometri.
2. Perlu dikembangakan metode yang dapat melakukan algoritma *blind watermarking* untuk citra *watermark* berwarna.

DAFTAR KODE SUMBER

BIODATA PENULIS

Mir'atul Mahmudah, lahir di Tulungagung pada tanggal 28 Juli 1992, merupakan anak pertama dari dua bersaudara. Penulis telah menempuh pendidikan mulai dari SDN Bungur I (1998-2004), SMPN 1 Kauman (2004-2007), SMAN 1 Kedungwaru (2007-2010) dan diterima sebagai mahaiswa Teknik Informatika ITS (2010-2014) melalui jalur beasiswa Bidik Misi. Selama masa kuliah, penulis pernah menjadi asisten pada mata kuliah Matematika Diskrit dan mata kuliah Komputasi Numerik. Selain itu, penulis juga aktif dalam beberapa organisasi di lingkungan ITS. Organisasi tersebut diantaranya Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) dan lembaga dakwah jurusan Keluarga Muslim Informatika (KMI). Penulis juga mengikuti beberapa kegiatan Latihan Keterampilan Manajemen Mahasiswa untuk tingkat pra Tingkat Dasar dan Tingkat Dasar. Dalam menyelesaikan kuliahnya, penulis mengambil bidang minat Komputasi Cerdas dan Visualisasi (KCV) dan tertarik pada hal yang berhubungan dengan citra dan data. Penulis dapat dihubungi melalui *e-mail* miratul.mahmudahh@gmail.com.