TUGAS AKHIR – KI141502

FORENSIK CITRA UNTUK ANALISIS COPY MOVE MENGGUNAKAN METODE EXHAUSTIVE SEARCH DAN AUTOCORRELATION

YUSUF NUGROHO
NRP 5112100147

Dosen Pembimbing I
Dr. Tohari Ahmad, S.Kom., MIT.

Dosen Pembimbing II
Hudan Studiawan, S.Kom., M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

IMAGE FORENSICS FOR COPY-MOVE ANALYSIS USING EXHAUSTIVE SEARCH AND AUTOCORRELATION METHOD

YUSUF NUGROHO
NRP 5112100147

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

Second Advisor
Hudan Studiawan, S.Kom., M.Kom.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2016

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

FORENSIK CITRA UNTUK ANALISIS *COPY-MOVE* MENGGUNAKAN *EXHAUSTIVE SEARCH* DAN *AUTOCORRELATION*

TUGAS AKHIR

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

Oleh:
YUSUF NUGROHO
NRP: 5112100147

Disetujui oleh Pembimbing Tugas Akhir:

1. Tohari Ahmad, S.Kom., MIT, PhD. .....
   (NIP. 197505252003121002) (Pembimbing 1)

2. Hudan Studiawan, S.Kom., M.Kom. .....
   (NIP. 198705112012121003) (Pembimbing 2)

SURABAYA
JUNI, 2016

(Halaman ini sengaja dikosongkan)

FORENSIK CITRA UNTUK ANALISIS *COPY-MOVE* MENGGUNAKAN *EXHAUSTIVE SEARCH* DAN *AUTOCORRELATION*

Nama Mahasiswa : YUSUF NUGROHO
NRP : 5112100147
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Tohari Ahmad, S.Kom., MIT,. Ph.D.
Dosen Pembimbing 2 : Hudan Studiawan, S.Kom., M.Kom.

Abstrak

Copy-move adalah suatu bentuk pemalsuan citra, dimana pengguna menyalin suatu bagian citra lalu menempelkannya di bagian lain pada citra yang sama. Teknik ini biasanya digunakan untuk menutupi objek dari suatu citra dengan cara menempelkan objek yang serupa di sekitarnya. Oleh karena itu diperlukan metode untuk mendeteksi serangan copy move, terlebih citra sering digunakan sebagai bukti dalam persidangan, dalam bidang kedokteran, dan bagian dari suatu berita.

Data citra seringkali terganggu dengan adanya noise, sehingga diperlukan metode filtering yang digunakan untuk meminimalisasi noise pada citra. Metode filtering dapat membantu proses pendeteksian serangan copy move pada citra. Oleh karena itu Tugas Akhir ini mengimplementasikan metode Gaussian dan Laplacian Filter sebagai penghilang noise pada citra yang akan dideteksi menggunakan metode exhaustive search dan autocorrelation.

Pada tugas akhir ini hasil yang didapat untuk menghilangkan noise menggunakan metode Gaussian Filter dianggap lebih efektif dibandingkan Laplacian Filter. Penggunaan kernel pada metode autocorrelation dianggap efektf dalam meningkatkan akurasi hasil deteksi serangan copy move pada citra. Sedangkan pada exhaustive search penggunaan kernel tidak diperlukan.

Kata kunci: Copy Move, Gaussian Filter, Laplacian Filter, Metode exhaustive search, autocorrelation, kernel morphology.

IMAGE FORENSICS FOR COPY-MOVE ANALYSIS USING EXHAUSTIVE SEARCH AND AUTOCORRELATION METHOD

Student's Name	**: YUSUF NUGROHO**
Student's ID	**: 5112100147**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Tohari Ahmad, S.Kom., MIT,. Ph.D.**
Second Advisor	**: Hudan Studiawan, S.Kom., M.Kom.**

Abstract

Copy-move forgery is a specific of image tempering where the part of image is copied and pasted in another part of the same image. This technique is usually used to cover up an objek in the image by attaching similar objek around it. Therefore developing method to verify and detect manipulated image became very important, especially because the image can be used as evidence in court, as a picture of headline news, and as a part of medical record.

The image data is often disturbed by a noise, therefore we need a filtering method that is used to minimize the noise. This filtering methods are useful to improve the copy move detection method. This study implements the method of Gaussian and Laplacian Filter to minimize the noise of image that will be detected using exhaustive and autocorrelation method.

In this undergraduate thesis, the results obtained using Gaussian Filter method to minimize noise was the effective way compared to Laplacian Filter. The use of kernel on autocorrelation method is considered effective for improving the accuracy of copy move image detection. Whereas on exhaustive Method, the use of kernel is not required.

Keywords : Copy Move, Gaussian Filter, Laplacian Filter, Metode exhaustive search, autocorrelation, kernel morphology.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah Swt yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**FORENSIK CITRA UNTUK ANALISIS COPY-MOVE MENGGUNAKAN METODE *EXHAUSTIVE SEARCH* DAN *AUTOCORRELATION***"

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Bapak Sudiro dan Ibu Cholifah selaku orang tua penulis yang selalu memberikan dukungan doa, moral, dan material yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Tohari Ahmad, S.Kom., MIT,. Ph.D. selaku pembimbing I yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
3. Bapak Hudan Studiawan, S.Kom., M.Kom. selaku II yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Bapak Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D. selaku dosen wali penulis yang telah memberikan arahan, masukan dan motivasi kepada penulis.
5. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknik Informatika ITS.
6. Bapak Prof.Ir. Supeno Djanali, M.Sc.,Ph.D sebagai dosen penguji Tugas Akhir penulis.

7. Bapak Dr. Radityo Anggoro, S.Kom.,M.Sc. selaku koordinator dan sebagai dosen penguji Tugas Akhir penulis.
8. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.
9. Ibu Eva Mursidah dan Ibu Sri Budiati yang selalu mempermudah penulis dalam peminjaman buku di RBTC.
10. Teman-teman Keluarga Muslim Informatika, yang sudah banyak meluruskan penulis.
11. Teman-teman seperjuangan RMK NCC/KBJ, yang telah menemani dan menyemangati penulis.
12. Teman-teman administrator NCC/KBJ, yang telah menemani dan menyemangati penulis selama penulis menjadi administrator, menjadi rumah kedua penulis selama penulis berkuliah.
13. Teman-teman angkatan 2012, yang sudah mendukung saya selama perkuliahan.
14. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2016

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

1.1 Latar Belakang

Berkembangnya kemampuan komputasi suatu komputer mempermudah seseorang untuk mendapatkan infomasi digital ditambah dengan banyaknya aplikasi dari pihak ketiga mempermudah seseorang untuk mengolah informasi yang didapat. Salah satu bentuk informasi yang mudah didapat adalah citra digital dan citra digital dapat dengan mudah diolah menjadi bentuk yang baru. Salah satu teknik digital untuk mengamati ada atau tidaknya suatu perubahan pada suatu citra digital adalah forensik citra digital yang memungkinkan suatu aplikasi dapat mengetahui adanya bentuk perubahan pada citra digital.

Terdapat banyak macam pemalsuan citra, salah satu diantaranya adalah *copy-move*. *Copy-move* adalah suatu cara dimana pengguna menyalin suatu bagian citra lalu menempelkannya di bagian lain pada citra yang sama. Teknik ini biasanya digunakan untuk menutupi objek dari suatu citra dengan cara menempelkan objek yang serupa pada bagian yang ingin ditutupi. Penelitian ini akan mendeteksi serangan *copy-move* dengan menggunakan metode *exhaustive search* dan *autocorrelation*. *Exhaustive search* berjalan dengan cara membentuk piksel-piksel citra menjadi *block-block* piksel berupa piksel tetangga lalu dilakukan operasi perbandingan.

Dengan melakukan implementasi algoritma *exhaustive search* dan *autocorrelation* maka diharapkan bisa melakukan analisa pada serangan citra digital yang menggunakan metode *copy-move* [1].

1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana melakukan deteksi terhadap serangan *copy-move*?
2. Apakah metode *exhaustive search* dan *autocorrelation* dapat digunakan untuk mendeteksi serangan *copy-move*?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut:

1. Bahasa pemrograman yang digunakan untuk menganalisa serangan *copy-move* ini adalah Python.
2. Dataset yang digunakan adalah citra digital yang memiliki kualitas rendah.
3. Jumlah data yang digunakan adalah 35 dan 20 data citra acak.
4. Metode yang digunakan untuk mendeteksi *copy-move* adalah metode *exhaustive search* dan *autocorrelation.*
5. Perangkat lunak yang digunakan adalah PyCharm sebagai IDE, Anaconda, dan basis data SQLite.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Dapat melakukan deteksi pemalsuan citra terhadap serangan *copy-move*.
2. Melakukan implementasi metode *exhaustive search* dan *autocorrelation.*

1.5 Manfaat

Dengan dibuatnya tugas akhir ini diharapkan dapat memberikan manfaat pada dunia forensik digital untuk mendeteksi pemalsuan citra terhadap serangan *copy-move*.

Sedangkan bagi penulis, tugas akhir ini bermanfaat sebagai sarana untuk mengimplementasikan ilmu dan algoritma

preprocessing image dan ilmu forensik yang telah dipelajari selama kuliah agar berguna bagi masyarakat.

1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal

Tahap awal tugas akhir ini adalah menyusun proposal tugas akhir. Pada proposal, diajukan gagasan untuk menganalisa dan mengidentifikasi pemasuan citra terhadap serangan *copy-move* menggunakan metode *exhaustive search* dan *autocorrelation*.

1.6.2 Studi Literatur

Pada tahap ini dilakukan untuk mencari informasi dan studi literatur apa saja yang dapat dijadikan referensi untuk membantu pengerjaan Tugas Akhir ini. Informasi didapatkan dari buku dan literatur yang berhubungan dengan metode yang digunakan. Informasi yang dicari adalah *exhaustive search*, *autocorrelation*, dan metode-metode *preprocessing image* seperti *Laplacian Filter* dan *Gaussian Filter.* Tugas akhir ini juga mengacu pada literatur jurnal karya Jessica Ftidrich, David Soukal, dan Jan Lukas dengan judul "*Detection of Copy-Move Forgery in Digital Images"* yang diterbitkan pada tahun 2003.

1.6.3 Implementasi Perangkat Lunak

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, maka dilakukan implementasi dengan menggunakan suatu perangkat lunak yaitu PyCharm sebagai IDE, Anaconda, dan basis data SQLite.

1.6.4 Pengujian dan Evaluasi

Pada tahap ini algoritma yang telah disusun diuji coba dengan menggunakan data uji coba yang ada. Data uji coba tersebut diuji coba dengan menggunakan suatu perangkat lunak dengan tujuan mengetahui kemampuan metode yang digunakan dan mengevaluasi hasil tugas akhir dengan jurnal pendukung yang ada. Hasil evaluasi juga mencakup kompleksitas dari kemampuan masing-masing metode tersebut dalam mendeteksi serangan *copy-move*.

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini. Secara garis besar, bab ini berisi tentang forensik digital, forensik citra digital, *copy-move*, SHA, Mean Squared Error dan SQLite.
3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi pembahasan mengenai perancangan dari metode *exhaustive search* dan *autocorrelation* yang digunakan untuk menedeteksi pemalsuan citra pada serangan *copy-move*.
4. Bab IV. Implementasi

Bab ini menjelaskan implementasi yang berbentuk *Pseudocode* yang berupa *Pseudocode* dari metode *exhaustive search* dan *autocorrelation.*

5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dari metode *exhaustive search* dan *autocorrelation* yang digunakan untuk menedeteksi pemalsuan citra pada serangan *copy-move* yang sudah diimplementasikan pada *Pseudocode*. Uji coba dilakukan dengan menggunakan dataset citra yang memiliki kualitas rendah. Hasil evaluasi mencakup kompleksitas dari kemampuan masing-masing metode tersebut dalam mendeteksi serangan *copy-move*.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat tabel-tabel data hasil uji coba dan *Pseudocode* program secara keseluruhan.

(Halaman ini sengaja dikosongkan)

BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam tugas akhir. Teori-teori tersebut diantaranya adalah *exhaustive search*, *autocorrelation,* dan beberapa teori lain yang mendukung pembuatan tugas akhir.

2.1 Forensik Digital

Forensik digital adalah aktivitas yang berhubungan dengan pemeliharaan, identifikasi, ekstraksi, dan dokumentasi bukti digital dalam kejahatan. Bukti-bukti dalam forensik digital sangatlah banyak, seperti semua bukti fisik (komputer, harddisk, dan lain lain), dokumen-dokumen yang tersimpan dalam komputer (gambar, pdf, video, dan lain lain), lalu lintas data dalam jaringan dan lain sebagainya. Karena luasnya cangkupan, forensik digital dibagi menjadi beberapa cabang seperti forensik komputer, forensik analisis data, forensik jaringan, forensik perangkat bergerak, forensik basis data, dan lain lain.

2.2 Forensik Citra Digital

Forensik citra digital merupakan cabang dari keamanan multimedia yang bertujuan untuk memvalidasi keaslian suatu citra dengan mengetahui informasi riwayatnya [2]. Beberapa kasus yang banyak terjadi dalam pemalsuan citra digital adalah *Copy-Splicing* dan *Copy-Move* dimana *Copy-Splicing* melibatkan dua citra atau lebih sedangkan *Copy-Move* hanya melibatkan satu citra. *Copy-Splicing* adalah memindahkan sebagian citra kepada citra lain sedangkan *Copy-Move* memindahkan sebagian citra pada bagian lainnya pada citra yang sama.

2.3 *Copy Move* Citra

Copy-move adalah sesuatu cara di mana pengguna menyalin sebagian citra lalu menempelkannya di bagian lain pada citra yang sama. Teknik ini biasanya digunakan untuk menutupi objek dari sebagian citra dengan cara menempelkan objek yang serupa di sekitarnya. Umumnya objek yang disalin adalah tekstur seperti rumput, dedaunan, kerikil, ataupun objek lain yang memiliki pola yang tidak beraturan tujuannya agar objek yang menutupi serupa dengan background atau objek di sekitarnya sehingga sulit dibedakan dengan mata.

Gambar 2.1 Contoh Copy-Move Citra

2.4 Citra Digital

Citra digital adalah gambar dua dimensi yang dihasilkan dari gambar *analog* dua dimensi yang *continue* menjadi gambar Diskrit melalui proses *sampling*. Gambar *analog* dibagi menjadi *N* baris dan *M* kolom sehingga menjadi gambar Diskrit. Persilangan antara baris dan kolom tertentu disebut dengan piksel. Contohnya adalah gambar/titik Diskrit pada baris *n* dan kolom *m* disebut juga piksel [*n, m*].

2.5 Anaconda

Anaconda adalah *open data science platform* yang dibangun menggunakan bahasa pemrograman Python. Versi Anaconda *open-source* menyediakan distribusi dengan performa tingkat tinggi dari bahasa pemrograman Python dan R yang berisikan lebih dari 100 paket untuk *data science.* Anaconda menyediakan CLI(*command line interface*) berupa perintah "conda" sebagai manajemen lingkungan untuk bahasa pemrograman Python. Perintah "conda" bisa untuk *create, export, list, remove* dan *update* lingkungan yang bisa digunakan untuk versi bahasa pemrograman Python atau paket yang dipasang didalamnya [3].

2.6 OpenCV

*Open*CV (*Open Source Computer Vision*) adalah *library* yang utamanya digunakan untuk pemrosesan visi komputer. *Open*CV adalah *library* gratis yang dapat digunakan di berbagai *platform*, seperti GNU/Linux maupun Windows. *Open*CV mulanya ditulis dalam basaha pemrograman C++, namun saat ini *Open*CV dapat digunakan pada berbagai bahasa seperti Python, Java, atau MATLAB [4].

2.7 NumPy

NumPy adalah paket dasar untuk komputasi ilmiah menggunakan Python yang berisi antara lain :

1. Objek *array* N dimensi
2. Fungsi yang mutakhir
3. Kakas bantu untuk mengintegrasi dengan *Pseudocode* C/C++ dan Fortran
4. Sangat berguna untuk aljabar linier, Fourier Transform, dan kemampuan angka *random*

Selain digunakan untuk hal ilmiah, NumPy juga bisa digunakan untuk *container* multidimensi yang efisien untuk data

generik. Tipe data *arbitrary* dapat didefinisikan, ini memungkinkan NumPy secara lancar dan cepat mengintegrasikan dengan banyak tipe basis data. NumPy adalah singkatan dari *Numeric Python* atau *Numerical Python*. Adalah sebuah modul *Open Source* untuk bahasa pemrograman Python yang menyediakan fungsi-fungsi yang telah *precompiled.* NumPy memperkaya kemampuan bahasa pemrograman Python dengan data struktur yang efisien untuk komputasi yang membutuhkan *arrays* atau *matrices* multidimensi [5].

2.8 SciPy

SciPy adalah singkatan dari *Scientific Python*. SciPy adalah modul *Library Open Source* yang digunakan untuk bahasa pemrograman Python. SciPy sering digunakan bersama dengan modul NumPy. SciPy menambah kemampuan NumPy dalam fungsi seperti *minimization, regression,* dan *Fast Fourier-transformation* [6].

2.9 SQLite

SQLite adalah mesin basis data SQL tertanam. Tidak seperti kebanyakan bais data SQL, SQLite tidak memiliki proses *server* yang terpisah. SQLite membaca dan menulis langsung kesebuah *file* kedalam *disk* penyimpanan. Sebuah basis data lengkap SQL berisi beberapa *table*, *indices, triggers,* dan *views* berada pada sebuah *file*. *File* basis data SQLite berupa *cross platform* yang bisa dengan mudah dipindah antara Sistem Operasi yang berarsitektur 32-bit dan 64-bit atau arsitektur *big-endian* dan *little-endian.* Karena fitur-fitur yang dimiliki oleh SQLite diatas maka membuat basis data SQLite sangat populer sebagai format *file* aplikasi [7].

2.10 SQLite3

SQLite3 adalah modul basis data yang sangat cepat dan sangat ringan untuk basis data SQLite. SQLite3 adalah modul *library* bahasa pemrograman Python yang digunakan untuk melakukan dengan basis data SQLite yang diletakan pada suatu *file* [8].

2.11 SHA

SHA adalah singkatan dari Secure *Hash* Algorithm. Keluarga algortima SHA dikembangkan oleh U.S National Institute of Standards and Technology (NIST). SHA adalah salah tipe *cryptographic* fungsi *Hash* yang menjamin tidak ada perubahan pada sebuah data. SHA menyelesaikan fungsi *Hash* dengan cara mengkomputasi nilai *Hash acrytographic* pada sebuah data. Perbedaan susunan data menghasilkan nilai *Hash* yang berbeda, *file* yang dimodifikasi bisa dibandingkan dengan nilai *Hash*nya dengan *file* aslinya. Nilai *Hash* menjamin integritas untuk setiap data yang diberikan kepada fungsi *Hash* karena dijamin setiap data memiliki nilai *Hash* yang berbeda [9].

2.12 Mean Squared Error

Mean Squared Error adalah metode untuk mengevaluasi dua buah matriks atau *array*. Masing-masing kesalahan atau sisa dikuadratkan. Kemudian dijumlahkan dan ditambahkan dengan jumlah observasi. Pendekatan ini mengatur kesalahan peramalan yang besar karena kesalahan-kesalahan itu dikuadratkan. Metode ini menghasilkan kesalahan-kesalahan sedang yang kemungkinan lebih baik untuk kesalahan kecil, tetapi kadang menghasilkan perbedaan yang besar [10].

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN PERANGKAT LUNAK

Bab ini membahas mengenai perancangan dan pembuatan sistem perangkat lunak. Sistem perangkat lunak yang dibuat pada tugas akhir ini adalah mendeteksi serangan *copy-move* pada citra digital dengan metode *exhaustive search* dan *autocorrelation.*

3.1 Data

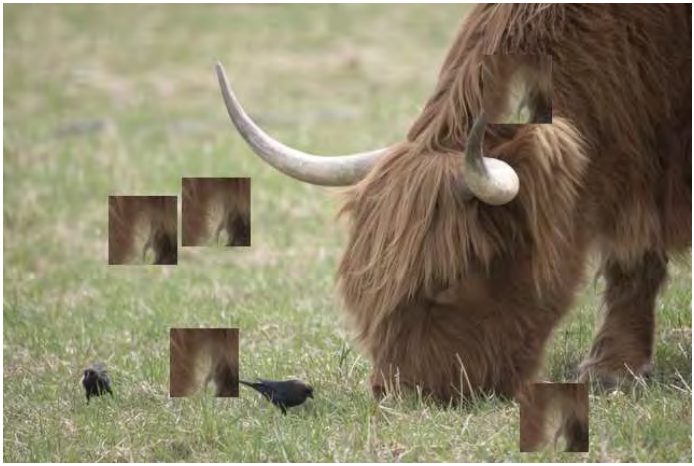
Pada sub bab ini akan dijelaskan mengenai data yang digunakan sebagai masukan perangkat lunak untuk selanjutnya diolah dan dilakukan pengujian sehingga menghasilkan data keluaran yang diharapkan.

3.1.1 Data Masukan

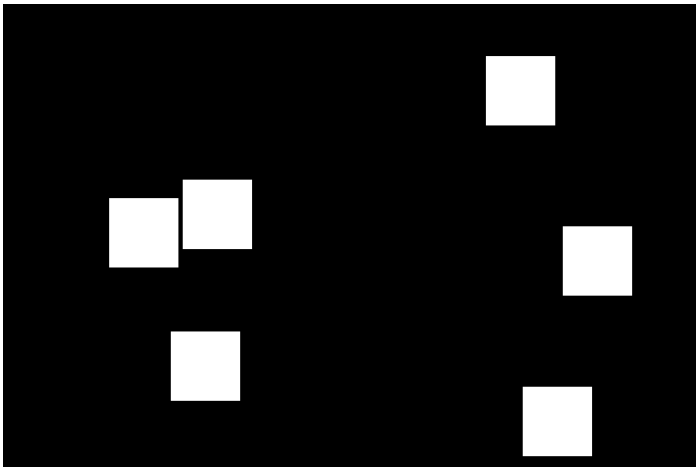
Data masukan adalah data yang digunakan sebagai masukan awal dari sistem. Data yang digunakan dalam perangkat lunak pendeteksi serangan *copy-move* pada citra digital terdiri dari dua jenis data masukan. Data masukan jenis pertama berjumlah 35 buah citra yang mengalami serangan *copy-move* dan 35 buah citra kunci jawaban yang menggambarkan bagian mana saja dari citra yang sesungguhnya terkena serangan *copy-move*. Data masukan jenis pertama yang berjumlah 35 citra tersebut digunakan sebagai data masukan untuk uji coba pada metode *exhaustive search* dan *autocorrelation*.

Sedangkan data masukan jenis kedua terdiri dari 20 buah citra yang mengalami serangan *copy-move* dan 20 buah citra kunci jawaban yang menggambarkan bagian mana saja dari citra yang sesungguhnya terkena serangan *copy-move*. Data masukan jenis kedua yang berjumlah 20 citra tersebut digunakan sebagai data masukan untuk uji coba pada modifikasi metode *exhaustive search* dan modifkasi metode *autocorrelation*.

. Contoh citra sebagai data masukan dan data kunci jawabannya ditunjukkan pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1 Contoh data masukan citra yang terkena serangan copy-move



Gambar 3.2 Contoh data masukan sebagai kunci jawaban dari hasil deteksi citra yang terkena serangan copy-move

3.1.2 Data Keluaran

Data masukan akan diproses dengan menggunakan metode *exhaustive search,* modifikasi metode *exhaustive search, autocorrelation*, dan modifikasi metode *autocorrelation.* Terdapat dua buah data masukan yaitu data citra dan data kunci jawaban. Hasil dari proses pendeteksian serangan *copy-move* pada masing-masing metode tersebut adalah suatu citra yang telah ditandai dengan kotak berwarna pada bagian yang terdeteksi terkena serangan *copy-move*.

3.2 Desain Umum Sistem

Rancangan perangkat lunak pendeteksi serangan *copy-move* pada citra menggunakan metode *exhaustive search*, dimulai dengan ekstraksi fitur citra menggunakan metode *Gaussian Filter* atau *Laplacian Filter*. Metode ini sangat membantu meminimalisasi *noise* yang ada pada suatu citra. Piksel-piksel pada citra yang telah diekstraksi menggunakan metode *Filter,* kemudian dikonversi ke dalam bentuk *Hash* menggunakan fungsi `SHA1`, lalu diurutkan (*sorting*) secara *ascending* (diurutkan berdasarkan nilai piksel yang paling kecil), sehingga piksel-piksel yang mirip akan berdekatan atau bersebelahan satu sama lain. Tahap selanjutnya adalah membandingkan tiap piksel dengan piksel yang ada disebelahnya. Tidak hanya satu piksel yang dibandingkan, namun ada 5 piksel tetangganya yang ikut dibandingkan. Sehingga apabila keenam piksel tersebut sama dengan keenam piksel lainnya, maka piksel-piksel tersebut lah yang terindikasi terkena serangan *copy-move*. Piksel-piksel yang sama tadi ditandai dengan warna yang berbeda. Diagram alir deteksi serangan *copy-move* pada citra menggunakan metode *exhaustive search*, ditunjukkan pada Gambar 3.4.

Metode *exhaustive search* ini dimodifikasi dengan menambahkan proses pengelompokkan piksel dan pengecekan

piksel-piksel yang teridentifikasi terkena serangan *copy-move* menggunakan nilai MSE dan *overlapping block*.

Metode kedua yang digunakan dalam perancangan perangkat lunak pendeteksi serangan *copy-move* pada citra adalah metode *autocorrelation*. Metode ini merupakan pengembangan dari metode sebelumnya, yaitu metode *exhaustive search*. Sehingga ada proses yang mirip dengan metode sebelumnya. Metode *autocorrelation* ini dimulai dengan mengekstrasi fitur citra dengan menggunakan metode *Gaussian Filter* atau *Laplacian Filter*. Metode ini juga sangat membantu meminimali *noise* yang ada pada suatu citra.

Tiap-tiap piksel pada citra yang telah diekstrasi menggunakan metode *Filter* sebelumnya, dihitung nilai autocorrelationnya menggunakan fungsi *autocorrelation*. Untuk mempercepat penghitungan fungsi *autocorrelation* tersebut, maka digunakan suatu algoritma *Fast Fourier Transform* atau yang biasa disingkat FFT. Algoritma *Fast Fourier Transform* adalah suatu algoritma untuk menghitung transformasi *Fourier* diskrit dengan cepat dan efisien. Transformasi Fourier diterapkan dalam beragam bidang, mulai dari pengolahan sinyal digital, memcahkan persamaan diferensial parsial, dan untuk algoritma untuk mengalikan bilangan bulat besar. Penerapan FFT dilakukan sebelum menjalankan proses *autocorrelation.*

Setelah nilai tiap piksel diganti dengan nilai hasil penghitungan *autocorrelation*, tahap selanjutnya adalah melakukan *sorting* terhadap piksel-piksel tersebut secara *descending* (diurutkan berdasarkan nilai piksel yang paling besar). Titik puncak pada grafik hasil penghitungan nilai *autocorrelation* dapat diindikasikan sebagai bagian dari citra yang terkena serangan *copy-move*. Bagian puncak tersebut dapat direpresentasikan sebagai nilai maksimum dari piksel-piksel hasil penghitungan *autocorrelation*.

Kemudian indeks dari piksel yang memiliki nilai maksimum tersebut dibandingkan dengan piksel sebelahnya menggunakan algoritma yang sama dengan metode *exhaustive search*, yaitu

membandingkan piksel tersebut dan 5 piksel tetangganya dengan piksel sebelahnya. Apabila keenam piksel tersebut sama dengan piksel sebelahnya maka piksel-piksel tersebut ditandai dengan warna yang berbeda. Berbeda dengan metode *exhaustive* sebelumnya, saat tahap membandingkan tidak dimulai dari indeks piksel yang pertama, tapi dari indeks piksel maksimum hasil penghitungan *autocorrelation*. Langkah ini diulangi lagi dengan mencari nilai maksimum selanjutnya, hingga *threshold* yang telah ditentukan tercapai. Diagram alir deteksi serangan *copy-move* pada citra menggunakan metode *autocorrelation* ditunjukkan pada Gambar 3.4.

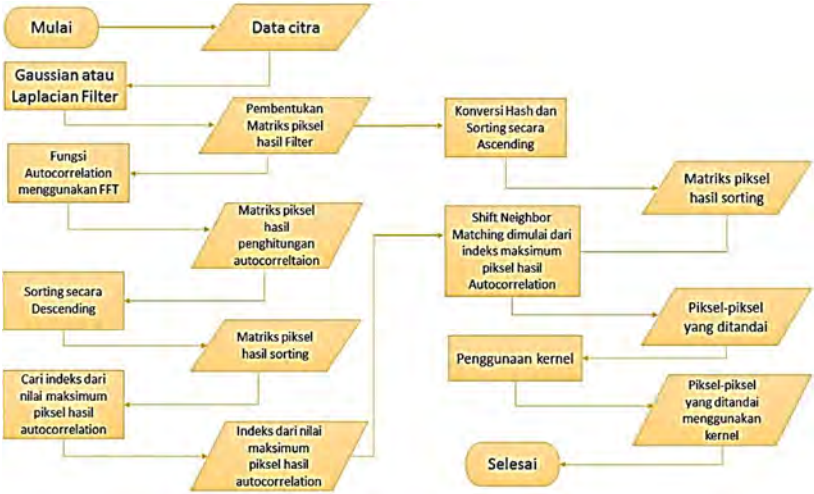
Metode *autocorrelation* ini dimodifikasi dengan menambahkan proses pengelompokkan piksel dan pengecekan piksel-piksel yang teridentifikasi terkena serangan *copy-move* menggunakan nilai MSE dan *overlapping block*.

Kernel morphology diterapkan pada proses deteksi serangan *copy-move* pada citra setelah proses dari kedua metode diatas selesai dijalankan. Dapat disimpulkan bahwa *kernel morphology* adalah metode yang digunakan untuk menyempurnakan hasil keluaran dari kedua metode sebelumnya. *Kernel morphology* digunakan saat pembentukan bagian-bagian pada piksel yang terdeteksi sebagai bagian dari *copy-move* citra. Bagian-bagian berupa *marked* pada piksel-piksel yang memenuhi ukuran *kernel* akan dibatasi dan ditandai dengan *kernel*, sehingga semua bagian (*marked*) yang masuk dalam ketentuan ukuran *kernel* akan digabungkan. Sebelum citra diproses menggunakan *kernel morphology*, citra akan diberikan *Gaussian Blur Filter* terlebih dahulu, karena untuk mendapatkan hasil *Canny Edge Detector* yang baik harus dilakukan *filter* tersebut.

Canny Edge Detector berfungsi untuk mendapatkan pola pinggiran pada objek hasil deteksi serangan *copy-move* yang nantinya akan diolah oleh beberapa *morphology kernel* untuk mendapatkan kemungkinan hasil deteksi serangan *copy-move.* Sebelum memproses citra hasil deteksi awal *copy-move*, terlebih dahulu diberikan perubahan pada citra dengan melakukan

transformasi citra yang berbasis RGB (*Red*, *Green*, dan *Blue*) menjadi *grayscale*. Citra yang sudah ditransformasi menjadi citra yang berupa *grayscale* akan mempermudah *Canny Edge Detector* mendapatkan pinggiran pada objek yang terdapat pada citra yang sudah ditandai dengan piksel putih ([255, 255, 255]) pada proses pendeteksian serangan *copy-move* sebelum diberikan *morphology kernel*. Citra dengan komposisi piksel *grayscale* akan memiliki kemungkinan kombinasi piksel yang lebih sedikit dibandingkan dengan citra yang memiliki komposisi RGB. Dengan itu maka proses *Canny Edge Detector* akan dengan lebih mudah mendapatkan batasan antara objek pada citra yang akan dibentuk menjadi bentuk objek yang berupa objek yang hanya memiliki pinggiran namun tidak memiliki isi.

Gambar 3.3 Proses deteksi *copy-move* citra menggunakan metode *exhaustive search*



Gambar 3.4 Proses deteksi *copy-move* citra menggunakan metode *autocorrelation*

3.3 *Exhaustive search*

Exhaustive search merupakan salah satu metode yang digunakan untuk mendeteksi serangan *copy-move* pada citra dengan cara membandingkan satu per satu piksel citra dengan piksel lainnya. Metode ini sangat mudah untuk diimplementasikan namun memiliki kompleksitas yang cukup tinggi. Sehingga diperlukan sedikit modifikasi pada metode ini agar dapat bekerja lebih efektif dan efisien.

Dimulai dengan membaca *file* citra dan mentransformasikannya dari bentuk RGB ke *integer*. Rumus konversi RGB ke *integer* pada suatu citra dapat dituliskan dengan Persamaan 3.1.

$$\mathrm{I} = (0{,}299 \times \mathrm{R}) + (0{,}587 \times \mathrm{G}) + (0{,}114 \times \mathrm{B}) \qquad (3.1)$$

Hasil konversi tiap-tiap piksel dari RGB ke *integer* tersebut diekstrasi menggunakan *Gaussian Filter* maupun *Laplacian Filter*.

Piksel-piksel pada citra yang telah diekstrasi menggunakan salah satu metode *Filter* tadi dihitung nilai *Hash* menggunakan fungsi `SHA1`. Nilai *Hash* memiliki nilai yang bersifat unik. Salah satu alasan menggunakan *Hash* dalam proses pengurutan yang akan dilakukan setelah ini adalah untuk memastikan bahwa piksel yang sama dan memiliki tetangga yang sama akan memiliki nilai *Hash* yang sama pula. Kemudian piksel-piksel yang telah di *Hash* diurutkan (*sorting*) secara *ascending* (diurutkan berdasarkan nilai piksel yang paling kecil), sehingga piksel-piksel yang mirip akan berdekatan atau bersebelahan satu sama lain.

Tahap selanjutnya adalah membandingkan nilai piksel dan tetangga piksel tersebut dengan nilai piksel disebelahnya beserta tetangga piksel disebelahnya. Algoritma *Shift Neighbor Matching* digunakan untuk mendeteksi piksel-piksel yang sama dan memiliki tetangga yang sama pula. Sehingga piksel-piksel yang teridentifikasi memiliki kemiripan yang sama dapat dipastikan piksel-piksel tersebut terkena serangan *copy-move*. Pada bagian citra yang terkena serangan *copy-move* tersebut ditandai dengan warna yang berbeda.

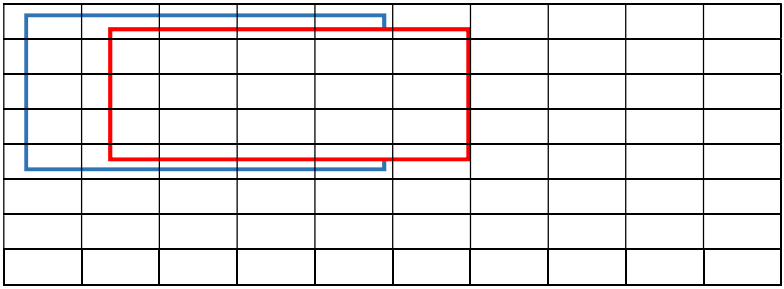
3.4 Modifikasi *Exhaustive Search*

Metode ini merupakan modifikasi dari metode *exhaustive search*. Pada metode *exhaustive search* yang sebelumnya proses pendeteksian *copy-move* dilakukan hanya dengan membandingkan satu piksel dengan piksel sebelahnya dengan menghitung kemiripan tetangganya menggunakan algoritma *neighbor shift matching*. Sedangkan modifikasi dilakukan dengan menambahkan proses pengelompokan piksel-piksel yang sama terlebih dahulu, sebelum diproses pada tahap pengecekan piksel menggunakan *neighbour shift matching*.

Dimulai dengan membaca *file* citra dan transformasi dari bentuk RGB ke bentuk *grayscale*. Rumus konversi RGB ke *grayscale*. Hal ini dilakukan untuk mempermudah konversi dari tiga nilai RGB ke dalam satu nilai *grayscale*.

Hasil konversi tiap-tiap piksel dari RGB ke *grayscale* tersebut diekstrasi menggunakan metode *Gaussian Filter* atau *Laplacian Filter*.

Tahap selanjutnya adalah melakukan *overlapping* citra dengan ukuran 10x10. *Overlapping* pada citra dapat digambarkan pada Gambar 3.5.



Gambar 3.5 Overlapping 10×10

Tiap kotak pada Gambar 3.5 diasumsikan memiliki nilai sebesar 2 piksel. Proses *overlapping* pertama pada area yang dibatasi garis berwarna hijau. *Overlapping* kedua berada pada area yang dibatasi garis berwarna merah. Banyaknya *Overlapping block* sebesar $L \times L$ pada citra berukuran $M \times N$ dapat dihitung pada Persamaan 3.2 dibawah ini.

$$(M - L + 1) \times (N - L + 1) \tag{3.2}$$

Sama seperti pada metode *exhaustive search*, piksel hasil ekstraksi beserta piksel-piksel tetangganya tadi di hitung nilai *Hash*nya menggunakan fungsi SHA1. Nilai *Hash* memiliki nilai yang bersifat unik. Salah satu alasan menggunakan *Hash* dalam proses pengurutan yang akan dilakukan setelah ini adalah untuk

memastikan bahwa piksel yang sama dan memiliki tetangga yang sama akan memiliki nilai *Hash* yang sama pula. Kemudian piksel-piksel yang telah di *Hash* tadi diurutkan (*sorting*) secara *ascending* (diurutkan berdasarkan nilai piksel yang paling kecil), sehingga piksel-piksel yang mirip akan berdekatan atau bersebelahan satu sama lain.

Selanjutnya piksel-piksel yang berdekatan tadi dikelompokkan berdasarkan kemiripan piksel yang sama. Sehingga proses pendeteksian piksel-piksel mana saja yang terdeteksi *copy-move* menjadi lebih efisien, karena proses pengecekan tidak perlu dilakukan pada setiap piksel pada citra tapi hanya dilakukan pada setiap anggota-anggota piksel pada *cluster*.

Pembentukan *cluster* dilakukan setelah proses *sorting*, yaitu dengan cara mengelompokkan piksel-piksel yang sama ke dalam satu *cluster*. Hal ini menjadi terlihat lebih mudah dan cepat karena sebelumnya piksel-piksel tersebut telah diurutkan secara *ascending* terlebih dahulu, sehingga piksel-piksel yang sama akan bersebelahan. Apabila piksel yang bersebelahan berbeda, maka kedua piksel tersebut terletak pada *cluster* yang berbeda. Proses pengelompokan ini digambarkan pada Gambar 3.6 dibawah ini.

Gambar 3.6 Pengelompokan piksel pada modifikasi metode *exhaustive search*

Deret piksel pada Gambar 3.6 tersebut merupakan gambaran piksel-piksel yang telah melewati tahap sorting sebelumnya, sehingga piksel-piksel yang sama akan bersebelahan satu sama lain. Proses pengecekan dilakukan dengan mendeteksi piksel disebelahnya apakah sama atau tidak. Apabila berbeda maka pengecekan anggota *cluster* berhenti.

Tahap selanjutnya pada masing-masing *cluster* dilakukan pengecekan piksel dengan memanfaatkan matriks 10×10 hasil *overlapping block* sebelumnya. Tiap anggota pada *cluster* dicek

dengan anggota lain pada *cluster* yang sama dengan menghitung nilai MSE antara kedua *overlapping block* tersebut. Pengecekan hanya dilakukan dengan setiap anggota pada satu *cluster* yang sama dan tidak dilakukan pengecekan dengan piksel-piksel lain yang bukan merupakan anggota piksel dari *cluster* tersebut.

Setelah didapatkan hasil MSE dari kedua block pada satu *cluster*, dilakukan pengecekan apakah nilai MSE tersebut lebih dari nilai *threshold* yang telah ditentukan. Nilai *threshold* yang digunakan pada modifikasi metode *exhaustive search* ini adalah sebesar **80**. Apabila nilai MSE kedua block pada proses pengecekan bernilai kurang dari **80,** maka kedua *block* tersebut dianggap sebagai piksel-piksel yang terkena serangan *copy-move*. Selanjutnya piksel-piksel yang teridentifikasi terkena serangan *copy-move* ditandai dan diberi warna yang berbeda.

3.5 *Autocorrelation*

Autocorrelation merupakan pengembangan dari metode sebelumnya, yaitu metode *exhaustive search*. Sehingga ada beberapa proses yang mirip dengan metode *exhaustive search*. *autocorrelation* merupakan proses konvolusi terhadap dirinya sendiri atau proses *correlation* dengan dirinya sendiri. Metode *autocorrelation* untuk mendeteksi serangan *copy-move* pada citra ini dimulai dengan mengekstrasi fitur citra dengan menggunakan metode *Laplacian Filter* atau *Gaussian Filter.* Sebelumnya piksel-piksel pada citra di konversi terlebih dahulu dari RGB ke *grayscale*.

Setiap piksel pada citra yang telah di *filter* menggunakan metode *Filter*, dihitung menggunakan fungsi *autocorrelation*. Persamaan fungsi *autocorrelation* ada pada Persamaan 3.3 dibawah ini.

$$r(k,l) = \sum_{i=0}^{M} \sum_{j=0}^{N} X(i,j) \times X(i+k, j+l)$$

(3.3)

Dengan nilai i,k = 0,...,*M*-1 dan nilai l,j = 0,...,*N*-1. Asumsi bahwa *X* merupakan citra yang telah di*filter* menggunakan metode *Filter* berukuran $M\ x\ N$. Untuk mempercepat penghitungan fungsi *autocorrelation* tersebut, maka digunakan suatu algoritma *Fast Fourier Transform* atau yang biasa disingkat FFT. Persamaan fungsi FFT apabila diterapkan pada metode *autocorrelation* dijelaskan pada persamaan 3.4.

$$R = F^{-}\{F(x) - F(\hat{x})\} \tag{3.4}$$

F merupakan fungsi `Fourier transform`.

Setelah nilai tiap piksel diganti dengan nilai hasil penghitungan *autocorrelation*, tahap selanjutnya adalah melakukan *sorting* terhadap piksel-piksel tersebut secara *descending* (diurutkan berdasarkan nilai *autocorrelation* yang paling besar). Pengurutan secara *descending* disebabkan titik puncak pada grafik *autocorrelation* dapat diindikasikan sebagai bagian dari citra yang terkena serangan *copy-move*. Bagian puncak tersebut dapat direpresentasikan sebagai nilai maksimum dari piksel-piksel hasil penghitungan *autocorrelation*.

Kemudian indeks dari piksel yang memiliki nilai maksimum tersebut dibandingkan dengan piksel sebelahnya menggunakan algoritma yang sama dengan metode *exhaustive search*, yaitu algoritma *neighbour shift matching*. Algoritma tersebut membandingkan piksel beserta 5 piksel tetangganya dengan piksel sebelahnya. Apabila keenam piksel tersebut sama dengan piksel sebelahnya maka piksel-piksel tersebut ditandai dengan warna yang berbeda. Berbeda dengan metode sebelumnya, saat tahap membandingkan tidak dimulai dari indeks piksel yang pertama, tapi dari indeks piksel maksimum hasil penghitungan *autocorrelation*. Langkah ini diulangi lagi dengan mencari nilai maksimum selanjutnya, hingga *threshold* yang telah ditentukan

tercapai. Selanjutnya piksel-piksel yang teridentifikasi terkena serangan *copy-move* ditandai dan diberi warna yang berbeda.

3.6 Modifikasi *Autocorrelation*

Metode ini merupakan modifikasi dari metode *autocorrelation* yang sebelumnya. Namun konsep modifikasi yang dilakukan pada metode ini hampir sama dengan modifikasi metode *exhaustive* sebelumnya.

Dimulai dengan membaca *file* citra dan transformasikannya dari bentuk RGB ke bentuk *grayscale*. Hasil konversi tiap-tiap piksel dari RGB ke *grayscale* tersebut diekstrasi menggunakan metode *Gaussian Filter* atau *Laplacian Filter*.

Tahap selanjutnya adalah melakukan *overlapping* citra dengan ukuran **10x10**. *Overlapping* pada citra digambarkan pada Gambar 3.5, yang telah dijelaskan pada modifikasi metode *exhaustive* sebelumnya. Hasil *overlapping block* citra ini disimpan untuk kemudian digunakan saat pengecekan nilai MSE.

Piksel-piksel yang telah dibaca sebelumnya dikonversi ke dalam bentuk *Hash* dan dihitung nilai *autocorrelation* menggunakan Persamaan 3.3. Untuk mempercepat penghitungan fungsi *autocorrelation* tersebut, maka digunakan suatu algoritma *Fast Fourier Transform* atau yang biasa disingkat FFT. Persamaan fungsi FFT dijelaskan pada Persamaan 3.4.

Setelah nilai tiap piksel diganti dengan nilai hasil penghitungan *autocorrelation*, tahap selanjutnya adalah melakukan *sorting* terhadap piksel-piksel tersebut secara *descending* (diurutkan berdasarkan nilai *autocorrelation* yang paling besar).

Selanjutnya piksel-piksel citra yang telah dikonversi kedalam bentuk *Hash*, kemudian diurutkan secara *ascending* lalu dikelompokkan berdasarkan kemiripan piksel yang sama. Berbeda dengan modifikasi metode *exhaustive search* sebelumnya, proses pengelompokkan pada piksel citra dilakukan dari indeks piksel

hasil nilai maksimum penghitungan *autocorrelation*. Proses pengelompokkan dijelaskan sebelumnya pada Gambar 3.6.

Lalu pada masing-masing *cluster* dilakukan proses pengecekan piksel-piksel yang teridentifikasi terkena serangan *copy-move* menggunakan nilai MSE pada tiap *overlapping block* yang dimiliki piksel tersebut. Tiap anggota pada *cluster* dicek dengan anggota lain pada *cluster* yang sama dengan menghitung nilai MSE antara kedua *overlapping block*. Pengecekan hanya dilakukan dengan setiap anggota pada satu *cluster* yang sama dan tidak dilakukan pengecekan dengan piksel-piksel lain yang bukan merupakan anggota piksel dari *cluster* tersebut.

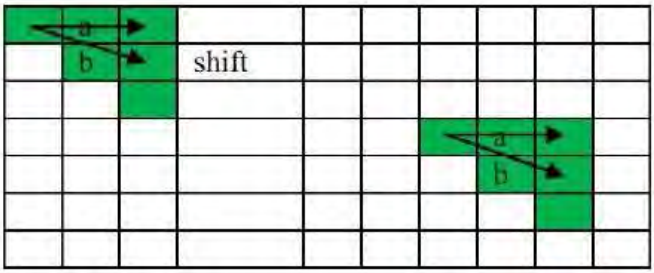
Setelah didapatkan hasil MSE dari kedua block pada satu *cluster*, dilakukan pengecekan apakah nilai MSE tersebut lebih dari nilai *threshold* yang telah ditentukan. Nilai *threshold* yang digunakan pada modifikasi metode *autocorrelation* ini adalah sebesar **150**. Apabila nilai MSE kedua block pada proses pengecekan bernilai kurang dari **150,** maka kedua *block* tersebut dianggap sebagai piksel-piksel yang terkena serangan *copy-move*. Selanjutnya piksel-piksel yang teridentifikasi terkena serangan *copy-move* ditandai dan diberi warna yang berbeda.

3.7 Algoritma Neighbor Shift Matching

Algoritma ini digunakan untuk mendeteksi piksel-piksel mana saja yang memiliki kemiripan yang sama dan dapat dipastikan bahwa piksel-piksel tersebut terkena serangan *copy-move*. Algoritma ini membandingkan tiap piksel dengan piksel yang ada disebelahnya. Tidak hanya satu piksel yang dibandingkan, namun ada 5 piksel tetangganya yang ikut dibandingkan. Sehingga apabila keenam piksel tersebut sama dengan keenam piksel lainnya, maka piksel-piksel tersebut lah yang terindikasi terkena serangan *copy-move*. Piksel-piksel yang sama tadi ditandai dengan warna yang berbeda.

Letak tetangga-tetangga suatu piksel yang akan dibandingkan dapat digambarkan pada Gambar 3.7. warna hijau

pada gambar didefinisikan sebagai nilai satu piksel. Apabila piksel yang berwarna hijau tersebut sama persis dengan piksel lain yang memiliki urutan *sequence* tetangga seperti pada Gambar 3.7, maka piksel-piksel tersebut ditandai dan diindikasikan sebagai piksel-piksel yang terkena serangan *copy-move*.



Gambar 3.7 Algoritma Neighbor Shift Matching

3.8 Metode *Filter*

3.8.1 Gaussian Filter

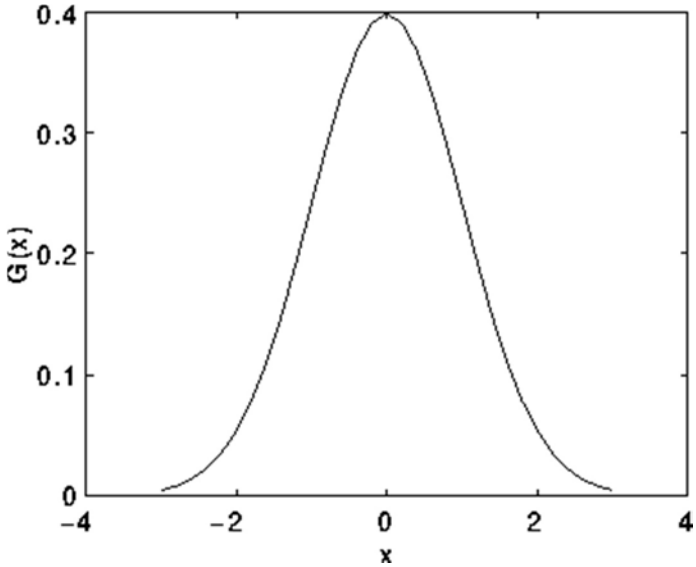
Gaussian Filter atau disebut juga *Gaussian Smoothing* adalah suatu metode yang digunakan untuk membuat efek *Blur* pada citra dan menghilangkan atau meminimalkan *noise* yang terdapat pada citra. Operator *Gaussian Filter* adalah operator dari Konvolusi 2-D. Ada dua persamaan distribusi *Gaussian Filter*, yaitu *Gaussian Filter* 1-D dan gaussian *filter* 2-D.

Pada metode *exhaustive search* ini persamaan yang digunakan adalah persamaan *Gaussian Filter* 1-D. Persamaan tersebut dapat dituliskan pada Persamaan 3.5.

$$G(x) = \frac{1}{\sqrt{2\sigma}}\, e^{\frac{x^2}{2\sigma^2}} \tag{3.5}$$

Nilai standar deviasi (σ) untuk *Gaussian Filter*, yang digunakan pada metode *exhaustive search* ini adalah **0,5.**

Apabila disribusi *Gaussian Filter* tersebut memiliki nilai *mean* (rata-rata) = 0, maka grafik distribusi *Gaussian Filter* dapat digambarkan seperti pada Gambar 3.8.



Gambar 3.8 Distribusi *Gaussian* dengan mean = 0 dan standar deviasi = 1

3.8.2 Laplacian Filter

Laplacian Filter adalah suatu metode yang digunakan untuk menghilangkan atau meminimalkan *noise* yang terdapat pada citra. *Laplacian* adalah sebuah penghitungan 2-D (dua dimensi) yang dihasilkan dari penurunan spasial dari sebuah citra digital. *Laplacian Filter* dari sebuah citra digital menyoroti bagian yang memiliki intensitas perubahan yang sangat berbeda dengan sekitarnya, maka *Laplacian Filter* sering digunakan untuk mendeteksi *edge* pada citra digital. *Laplacian Filter* sering

diterapkan untuk sebuah citra digital yang sebelumnya sudah dihaluskan dengan diberikan *filter Gaussian smoothing filter* yang berguna untuk mereduksi sensitifitas *Laplacian Filter* pada *noise* yang terdapat pada citra digital.

Laplacian Filter sendiri merupakan turunan dari *Gaussian Filter*. Persamaan dari fungsi *Laplacian Filter* dituliskan pada persamaan 3.6.

$$L(x,y) = \nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \qquad (3.6)$$

3.8.3 Gaussian blur

Gaussian Blur adalah *filter* yang menempatkan warna transisi yang signifikan dalam sebuah citra, kemudian membuat warna-warna pertengahan untuk menciptakan efek *blur* pada sisi-sisi sebuah citra.*Gaussian Blur* adalah salah satu *filter* blur yang menggunakan fungsi *Gaussian* untuk menciptakan efek *autofocus* untuk mengurangi detail dan menciptakan efek berkabut.

Gaussian Blur sendiri merupakan konvolusi citra dengan fungsi *Gaussian Blur*. *Gaussian Blur* digunakan sebagai tahap pertama dari algoritma deteksi tepi *Canny*. Untuk menggunakan gaussian smoothing, penulis perlu melakukan proses konvolusi. Konvolusi merupakan proses perjumlahan seluruh hasil perkalian antara matriks *filter* dengan matriks perluasan tetangga dari titik (*x, y*) pada citra.

3.9 Kernel Morphology

Proses *morphology* digunakan untuk menghilangkan ketidaksempurnaan bentuk yang ada dalam suatu citra. Dengan operasi *morphology Canny edge detector, closing,* dan *opening*, kombinasi ketiganya dapat menghasilkan citra yang lebih sesuai dengan jawaban yang diinginkan.

Morphology Canny edge detector dikenal juga sebagai *optimal detector*. Algoritma *Canny* bertujuan untuk memenuhi 3 kriteria:

1. *Low error rate*
   Sebuah pendeteksian yang baik untuk mendeteksi *edge* yang sudah ada.
2. *Good localization*
   Jarak antara tepi piksel yang asli dan tepi piksel yang dideteksi harus memiliki perbedaan yang sangat minimum
3. *Minimal response*
   Hanya satu respon detektor pada setiap tepi.

Morphology opening adalah nama lain dari tahap erosi yang diikuti langsung dengan proses dilasi. Sangat berguna untuk mengurangi atau menghilangkan *noise*.

Morphology closing adalah kebalikan dari *opening morphology*. Dilasi diikuti dengan erosi sangat berguna pada saat menutup lubang kecil di dalam *objek foreground* atau sebuah poin kecil yang hitam pada sebuah objek.

Ide dasar dari *morphology* erosi adalah seperti erosi yang terjadi pada tanah, proses ini mengerosikan batasan pada *objek foreground. Kernel* bergeser pada citra pada konvolusi 2-D. Sebuah piksel pada citra asli (baik satu ataupun nol) akan dianggap satu hanya jika semua piksel dibawah *kernel* adalah satu, selain itu dierosikan menjadi nol. Semua piksel yang dekat dengan batas pada *objek foreground* akan diabaikan tergantung pada ukuran *kernel*. Sehingga ketebalan atau ukuran pada *objek foreground* berkurang atau bagian / daerah yang putih berkurang pada citra. Ini sangat berguna untuk mengurangi atau menghilangkan *noise* putih yang kecil.

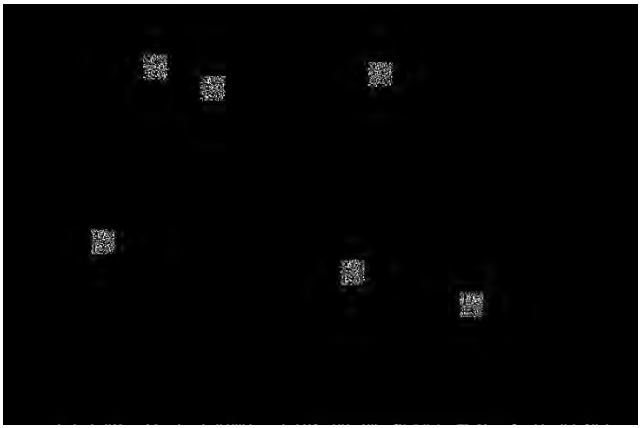
Dilasi adalah kebalikan dari erosi. Karena *noise* yang sudah hilang atau berkurang karena *morphology opening*, Area pada *objek foreground* bertambah. Ini sangat berguna untuk menggabungkan bagian yang terputus dari sebuah objek.

Morphology kernel digunakan untuk meningkatkan akurasi sistem dalam mendeteksi serangan *copy-move* pada citra. Sebelum implementasi *Morphology kernel* dilakukan, citra hasil keluaran dari metode-metode yang dijelaskan sebelumnya dikonversi terlebih dahulu ke dalam bentuk grayscale. Hal ini dilakukan karena hasil keluaran dari penggunaan *Morphology kernel* adalah citra berwarna hitam putih. Warna hitam sebagai background, sedangkan warna putih menandai area mana saja yang terkena serangan *copy-move*. Selanjutnya citra hasil konversi *grayscale* tersebut di-*filter* menggunakan metode *Gaussian blur*.

Ada 3 jenis *Morphology kernel* yang digunakan dalam proses pendeteksian serangan *copy-move* pada citra:

1. *Kernel edge*

Konsep dasar dari *kernel edge* adalah memberikan garis tepi terhadap sekumpulan piksel-piksel yang berada pada area yang ditentukan pada *threshold*. Misal apabila *threshold* yang digunakan adalah 20×20. Maka piksel-piksel yang terdeteksi berada pada area 20×20 akan diberikan garis tepi, untuk menandai bagian mana saja pada citra yang terdeteksi terkena serangan *copy-move*. Gambar 3.9 adalah hasil keluaran dari penggunaan *kernel edge* pada citra yang terdeteksi terkena serangan *copy-move*.

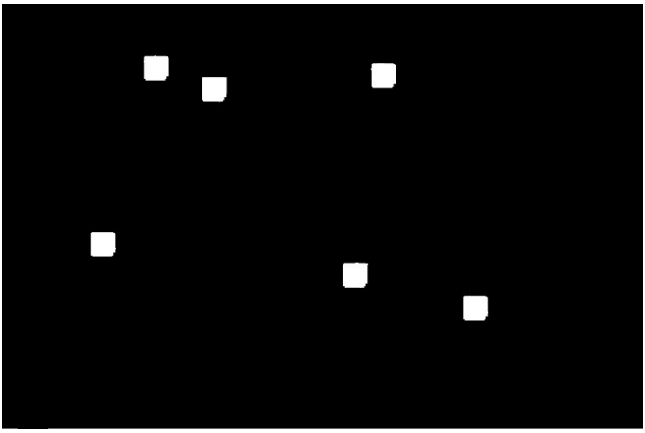


Gambar 3.9 Contoh hasil keluaran *Kernel Egde*

2. *Kernel Closing*

Kernel closing menghasilkan *output* yang baik jika digabungkan dengan penggunaan *kernel edge*. Pada hasil keluaran *kernel edge*, pada area yang diberi garis tepi, bagian dalam dari area tersebut umumnya terdiri dari titik-titik warna yang menandai piksel-piksel yang terdeteksi. Titik-titik warna tersebut dihilangkan dan diganti dengan satu warna utuh yang memenuhi area hingga batas tepi dari *kernel edge*. Sehingga hasil keluaran yang dihasilkan lebih terlihat akurat dan jelas. Gambar 3.10 Adalah hasil keluaran dari penggunaan *kernel closing* pada Gambar 3.9 Yang merupakan hasil keluaran dari *kernel edge*.

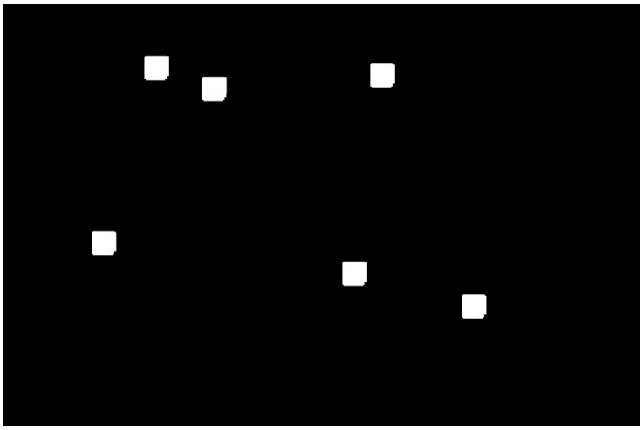
Pada gambar 3.10 terlihat bahwa hasil keluaran citra yang tadinya berupa titik-titik pada Gambar 3.9, yang merupakan hasil keluaran *kernel edge*, berubah menjadi kotak dengan warna putih yang memenuhi seluruh bagian dalam dari tepi-tepi yang telah dihasilkan oleh *kernel edge* sebelumnya.



Gambar 3.10 Contoh hasil keluaran *Kernel Closing*

3. *Kernel Opening*

Kernel opening mewarnai area yang telah ditentukan sebelumnya. Misalnya ukuran *kernel opening* adalah 10×10, apabila ada area yang ditandai terkena serangan *copy-move* sebesar 10×10, maka area tersebut akan diberi warna oleh *kernel opening*. Bila ukuran objek yang tidak memenuhi kriteria yang telah diatur maka akan ditandai sebagai objek *background* atau dikonversi warna pikselnya. *Kernel opening* dapat mengurangi jumlah *noise*yang dihasilkan oleh *kernel opening* sebelumnya, dikarenakan bentuk *noise* yang cenderung tidak memenuhi kriteria ukuran minimun yang telah ditentukan. Ukuran pada *kernel opening* dapat menentukan jumlah *noise*, jika ukuran *kernel opening* diatur besar maka akan memperkecil kemungkinan terdapatnya *noise* namun dapat meningkatkan kemungkinan penurunan akurasi hasil deteksi serangan *copy-move.* Contoh hasil keluaran dari *kernel opening* ada pada Gambar 3.11.



Gambar 3.11 Contoh hasil keluaran *kernel opening*

3.10 Basis Data

Rancangan Basis data pada sistem untuk mendeteksi serangan *copy-move* pada citra terdiri dari satu entitas yaitu *Tabel_Analisis*.

Data hasil deteksi serangan *copy-move* pada citra tiap metode disimpan ke dalam entitas *Tabel_Analisis* pada basis data. Data pada basis data digunakan untuk melihat hasil perangkat lunak pendeteksi serangan *copy-move* pada citra dan dapat digunakan untuk analisa setiap algoritma yang digunakan untuk mendeteksi serangan copy-move pada citra. Analisa pada basis data digunakan untuk melihat performa pada setiap algoritma yang digunakan yang nantinya akan digunakan untuk diambil kesimpulan. Atribut-atribut yang terdapat pada entitas *Tabel_Analisis* dijelaskan pada Tabel 3.1.

Tabel 3.1 Atribut-atribut pada entitas Tabel_Analisis

No	Atribut	Penjelasan
1	ID	Id Citra (*Increment*)

2	Nama_*File*	Nama *file* citra
3	*Kernel*_X_*opening*	Ukuran *kernel* X *opening*
4	*Kernel*_Y_*opening*	Ukuran *kernel* Y *opening*
5	*Kernel*_X_*closing*	Ukuran *kernel* X *closing*
6	*Kernel*_Y_*closing*	Ukuran *kernel* X *closing*
7	*Canny*_X	Ukuran kernel X *Canny edge detector*
8	*Canny*_Y	Ukuran kernel Y *Canny edge detector*
9	Nilai_Tetangga	Jumlah tetangga piksel
10	MSE_Un*morphology*	Nilai MSE tanpa *kernel*
11	MSE_*opening*_Morphology	Nilai MSE *kernel opening*
12	MSE_*closing*_Morphology	Nilai MSE *kernel closing*
13	MSE_Black	Nilai MSE citra *black*
14	Date	Tanggal dan waktu uji coba
15	Status	Status metode *Filter* yang digunakan
16	Time Calculation	Durasi eksekusi

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Bab ini berisi penjelasan mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa *Pseudocode* untuk membangun program.

4.1 Lingkungan Implementasi

Implementasi pendeteksian serangan *copy-move* pada citra dengan metode *exhaustive search* dan *autocorrelation* menggunakan spesifikasi perangkat keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Perangkat Lunak

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel(R) Core(TM) i5-2.6 GHz
	Memori	10 GB 1600 MHz DDR3
Perangkat Lunak	Sistem Operasi	Windows 8 dan Ubuntu 14.04
	Perangkat Pengembang	PyCharm Ultimate and Community Edition dan Sqlitebrowser

4.2 Implementasi

Pada sub bab implementasi ini menjelaskan mengenai pembangunan perangkat lunak secara detail dan menampilkan *Pseudocode* yang digunakan mulai tahap *preprocessing* hingga pendeteksian citra menggunakan metode *exhaustive search* dan *autocorrelation*. Pada tugas akhir ini data yang digunakan, seperti yang telah dijelaskan di bab sebelumnya, yaitu terdiri dari dua jenis data masukan. Data masukan jenis pertama berjumlah 35 buah citra

yang mengalami serangan *copy-move* dan 35 buah citra kunci jawaban yang menggambarkan bagian mana saja dari citra yang sesungguhnya terkena serangan *copy-move*. Data masukan jenis pertama yang berjumlah 35 citra tersebut digunakan sebagai data masukan untuk uji coba pada metode *exhaustive search* dan *autocorrelation*. Sedangkan data masukan jenis kedua terdiri dari 20 buah citra yang mengalami serangan *copy-move* dan 20 buah citra kunci jawaban yang menggambarkan bagian mana saja dari citra yang sesungguhnya terkena serangan *copy-move*. Data masukan jenis kedua yang berjumlah 20 citra tersebut digunakan sebagai data masukan untuk uji coba pada modifikasi metode *exhaustive* dan *autocorrelation* modifikasi.

4.2.1 Metode *Exhaustive search*

Hal pertama yang dilakukan pada Metode *exhaustive search* ini adalah menginisialisasi variabel-variabel yang akan digunakan dalam proses pendeteksian serangan *copy-move* terhadap citra melalui *Pseudocode* 4.2. Inisialisasi dilakukan dengan menyiapkan variabel-variabel *array* yang digunakan dalam proses pendeteksian dan *path* mana yang akan digunakan untuk membaca *file-file* citra sebagai data masukan dari sistem. Setelah mendapatkan nilai setiap piksel pada *array* maka langkah selanjutnya adalah membentuk *neighbour shift matching* yang akan mempermudah untuk mengurutkan nilai piksel yang mempunyai kemiripan. Setelah diurutkan dengan pola *neighbour shift matching* akan dibentuk *cluster* yang akan memiliki kemiripan satu dengan yang lain. Untuk nilai yang memiliki kemiripan total dengan indeks setelahnya akan ditandai sebagai piksel yang terkena serangan *copy-move*. Bila piksel yang sedang dibandingkan dengan indeks setelahnya tidak memiliki kemiripan maka piksel tersebut akan dibandingkan dengan indeks yang sebelumnya, bila memiliki kemiripan total maka akan ditandai sebagai piksel yang terkena serangan *copy-move.* Pencocokan

dilakukan menggunakan nilai *hash* yang dibentuk dari nilai piksel dan nilai-nilai *neighbour shift matching*.

```
1   CLASS ExhaustiveMethod(object):
2       FUNCTION __init__(self):
3             boxCompareResult <- []
4             array_filter <- []
5             boxFilter <- []
6             Nama_File <- 0
7             Kernel_X <- 0
8             Kernel_Y <- 0
9             Nilai_Tetangga <- 0
10            MSE <- 0
11            MSE_BLACK <- 0
12            DATE <- 0
13      ENDFUNCTION
14
15      FUNCTION initPath(self,pathNamaFile,
        pathHasilFile, pathKunciJawaban,
        pathMask):
16            pathNamaFile <- pathNamaFile
17            pathHasilFile <- pathHasilFile
18            pathKunciJawaban<- pathKunciJawaban
19            pathMask <- pathMask
20            matriceOfSourceImage <- []
21      ENDFUNCTION
```

Pseudocode* 4.1 Inisialisasi variabel yang digunakan pada metode *Exhaustive Search

Tahap selanjutnya adalah membaca piksel-piksel tiap citra dan dikonversi menjadi bentuk *integer* melalui *Pseudocode* 4.2. Konversi yang dilakukan pada piksel digunakan untuk mempermudah membandingkan antara piksel dibandingkan dengan membandingkan bentuk piksel asli yang menggunakan tiga *variable* RGB.

```
1   FUNCTION rgbToInt(self, rgb):
2           RETURN rgb[0] * 65536 + rgb[1] * 256
            + rgb[2]
3   ENDFUNCTION
4
5   FUNCTION           loadThe FileUp           (self,
    pathOriginalImage File):
6           listGlob <- pathOriginalImage File
7           file Name <- listGlob
8           file Temp <- file Name.split('.')
9           file Type <- "." + file Temp[1]
10          file Name <- file Temp[0]
11          file Type <-  file Type
12          file Name <-  file Name
13          imageSource File FullPath <-
            pathNama File+ file Name + file Type
14          imageSource <-
            cv2.imread(imageSource File FullPath)
15          height,width,channels <-
            imageSource.shape
16          height <- height
17          width <- width
18          channels <- channels
19          OUTPUT "Working on file \t\t\t: " +
            imageSource File FullPath
20  ENDFUNCTION
```

***Pseudocode* 4.2 Kode Program membaca piksel citra dan konversi piksel citra ke integer**

Langkah pertama pada *Pseudocode* 4.2 adalah membaca piksel-piksel citra dengan menggunakan *library cv2* pada *morphology opening* CV. Piksel-piksel citra tersebut disimpan ke dalam sebuah *array* yang selanjutnya dikonversikan ke dalam bentuk *integer* menggunakan Persamaan 4.1. Persamaan dibawah ini akan mengubah setiap elemen piksel yang berbentuk RGB *array* yaitu piksel *Red*, *Green*, dan *Blue*.

$$\mathrm{I} = (0{,}299 \times \mathrm{R}) + (0{,}587 \times \mathrm{G}) + (0{,}114 \times \mathrm{B}) \qquad (4.1)$$

Pada metode *exhaustive search*, hasil dari konversi piksel-piksel tersebut diminimalisasi jumlah *noise*nya menggunakan metode *Gaussian Filter* melalui *Pseudocode* 4.3 atau menggunakan metode *Laplacian Filter* melalui *Pseudocode* 4.4.

```
1   FUNCTION gaussian(self):
2       height <-  height
3       width <-  width
4       imgSource <-  imageSource
5       intImageArray <- np.zeros((height,
        width,), dtype=np.int)
6       for i in range(0, height):
7           for j in range(0, width):
8               intImageArray[i][j] <- (
9               rgbToInt(imgSource[i, j]))
10          ENDFOR
11      ENDFOR
12      scipy.ndimage.filters.gaussian_filter
        (intImageArray,0.5,order=0,output=
        array_filter, mode='reflect',cval=0.0,
        truncate = 4.0)
13      for sumbuX in range(0, height):
14          for sumbuY in range(0, width):
15               boxFilter.append([sumbuX,
                 sumbuY,array_filter
                 [sumbuX][sumbuY]])
16          ENDFOR
17      ENDFOR
18      boxFilter.sort
        (key=operator.itemgetter(2),
        reverse=True)
19  ENDFUNCTION
```

Pseudocode* 4.3 Kode Program *Gaussian Filter

Implementasi metode *Gaussian Filter* pada *Pseudocode* 4.3 menggunakan *library scipy.ndimage* dengan memasukkan nilai standar deviasi sebesar 0,5.

```
1   FUNCTION Laplacian(self):
2       height <-  height
3       width <-  width
4       intImageArray <- []
5       imgSource <-  imageSource
        intImageArray <- np.zeros((height,
6       width,), dtype=np.int)
7       imgSource<- cv2.GaussianBlur(imgSource,
8       (5, 5), 0)
9       for i in range(0, height):
10          for j in range(0, width):
11              intImageArray[i][j]<- (
12              rgbToInt(imgSource[i, j]))
            ENDFOR
        ENDFOR
        scipy.ndimage.filters.laplace
13      (intImageArray, output= array_filter,
        mode='reflect', cval=0.0)
        for sumbuX in range(0, height):
14          for sumbuY in range(0, width):
15               boxFilter.append([sumbuX,
16               sumbuY,  array_filter
                 [sumbuX][sumbuY]])
            ENDFOR
17      ENDFOR
18      boxFilter.sort(
19      key=operator.itemgetter(2),
        reverse=True)
20  ENDFUNCTION
```

***Pseudocode* 4.4 Kode Program *Laplacian* Filtering**

Implementasi metode *Laplacian Filter* pada *Pseudocode* 4.4 menggunakan *library scipy.ndimage*.

Selanjutnya hasil *Filtering* tiap piksel tersebut dan kelima piksel-piksel tetangganya(sesuai dengan algoritma *neighbour shift matching*) dimasukkan ke dalam sebuah *array* dan akan dilakukan pengurutan setelah diproses dengan parameter nilai pada *array* indeks kedua.

```
1   FUNCTION makeAttributeBox(self):
2       height <-  height
3       width <-  width
4       imageSource <-  array_filter
5       for sumbuX in range(0, height):
6           for sumbuY in range(0, width):
7               value <- imageSource[sumbuX, sumbuY]
                IF (sumbuX + 1 <= height - 1):
8                   A <- np.array (imageSource
9                   [sumbuX + 1, sumbuY])
10              ELSE:
11                  A <- 0
12              ENDIF
13              IF (sumbuY + 1 <= width - 1 AND
14              sumbuX + 1 <= height - 1):
                    B <- np.array (imageSource
15                  [sumbuX + 1, sumbuY + 1])
                ELSE:
16                  B <- 0
17              ENDIF
18              IF (sumbuX + 2 <= height - 1):
19                  A2 <- np.array (imageSource
                    [sumbuX + 2, sumbuY])
                ELSE:
20                  A2 <- 0
21              ENDIF
22              IF (sumbuY + 1 <= width - 1 AND
23              sumbuX + 2 <= height - 1 AND
                sumbuX <= height):
                    B2 <- np.array (imageSource
24                  [sumbuX + 2, sumbuY + 1])
                ELSE:
25                  B2 <- 0
26              ENDIF
27              IF (sumbuY + 2 <= width - 1 AND
28              sumbuX + 2 <= height - 1):
                    XX <- np.array (imageSource
29                  [sumbuX + 2, sumbuY + 2])
                ELSE:
30                  XX <- 0
31              ENDIF
32              temporaryNeighbor <-
33              np.hstack (( value, A, A2, B, B2,
                XX))
```

```
                hasil Hash <-  Hash
34              (temporaryNeighbor)

35  matriceOfSourceImage.append([sumbuX,      sumbuY,
36  value, A, A2, B, B2, XX, hasilHash])
37  ENDFUNCTION
```

***Pseudocode* 4.5 Kode Program pembuatan array yang berisi atribut piksel tetangga**

Pada *Pseudocode* 4.5 hal yang dilakukan adalah mengambil sebuah piksel beserta kelima piksel-piksel tetangganya. Apabila piksel tetangga tersebut berada pada pojok maupun tepi citra maka piksel – piksel tetangga disebelah kanan, atas, kiri, maupun bawah yang sudah melewati batas ukuran citra, di-*set* menjadi 0. Setelah itu hasil dari piksel dan piksel-piksel tetangganya dikonversi ke dalam bentuk *Hash* melalui *Pseudocode* 4.6.

```
1   FUNCTION Hash(self, array):
2       HasilHash <- Hashlib.sha1(array).hexdigest()
3       RETURN hasilHash
4   ENDFUNCTION
```

***Pseudocode* 4.6 Kode program fungsi Hash**

Fungsi *Hash* tersebut digunakan untuk membuat unik nilai dari suatu piksel dan piksel-piksel tetangganya, sehingga meningkatkan akurasi pengecekan kemiripan antar *block* piksel. *block* piksel disini merupakan piksel-piksel tetangga yang digabungkan menjadi satu pada *Pseudocode* 4.5 sebelumnya.

Selanjutnya hasil dari konversi piksel dan piksel-piksel tetangga ke dalam bentuk *Hash*, disimpan ke dalam sebuah *array*. Array tersebut berisi letak/indeks piksel (sumbu *x*, sumbu *y*), nilai piksel dan tetangga piksel, serta nilai hasil *Hash* piksel–piksel tersebut.

Tahap selanjutnya adalah mengelompokkan piksel-piksel yang sama. Pengelompokan piksel pada citra dilakukan dengan men*sorting* piksel-piksel citra yang telah mengalami *preprosessing* secara *ascending* melalui *Pseudocode* 4.6.

```
1   FUNCTION Sort_ImgVector(self):
2       matriceOfSourceImage.sort
        (key = operator.itemgetter(8))
3   ENDFUNCTION
```

***Pseudocode* 4.7 Kode Program mensorting piksel**

Proses pengurutan piksel-piksel citra pada *Pseudocode* 4.7 di atas menggunakan *library sort* pada Python. Sehingga piksel-piksel yang identik akan diletakkan bersebelahan dan berdekatan satu sama lain setelah dirutkan secara ascending.

Setelah piksel-piksel yang identik diletakkan bersebelahan/berdekatan satu sama lain pada proses *sorting*, maka selanjutnya adalah membandingkan piksel-piksel tersebut dengan piksel sebelahnya. Pixel-piksel tetangga pada piksel tersebut juga dibandingkan dengan piksel-piksel tetangga yang dimiliki piksel sebelahnya menggunakan algoritma *neighbour shift matching* melalui *Pseudocode* 4.7. Pengurutan nilai *Hash* neighbour shift matching akan lebih cepat dan menghasilkan nilai yang unik pada setiap kombinasi antara piksel dan tetangganya, maka metode pengurutan dengan menggunkan *Hash* akan menjamin nilai yang berdekatan adalah nilai yang berisikan/beranggotakan kumpulan piksel yang mirip. Karena setiap kombinasi yang dikonversi menjadi nilai *Hash* akan menghasilkan nilai yang berbeda. Pada penerapan metode konversi nilai setiap piksel dengan tetangganya akan membutuhkan kompleksitas yang lebih rendah ketika membandingkan antara setiap piksel dan anggota *neighbour shift matching* dengan nilai piksel selanjutnya yang juga diikuti dengan nilai *neighbour shift matching*. Oleh karena itu maka konversi setiap nilai piksel dan anggota *neighbour shift matching* menjadi nilai Hash akan mempermudah dalam pada proses pengurutan dan pada proses pencocokan piksel.

```
1   FUNCTION CheckShiftVector(self):
2        boxCompareResult <- []
3        matriceOfSourceImage <-
         matriceOfSourceImage
4        for eachItem in range(0,
         len(matriceOfSourceImage)):
5           nilaiTreshold <- 0
6           IF (eachItem + 1 <=
            len(matriceOfSourceImage) - 1):
7               IF (np.array_equal(
                matriceOfSourceImage[eachItem][2],
                matriceOfSourceImage[eachItem+1[2])
                AND
               (matriceOfSourceImage[eachItem][8]=
               matriceOfSourceImage
               [eachItem+1][8])):
8                    for neighborIteration in
                     range(3,8):
9                      IF(np.array_equal
                       (matriceOfSourceImage
                       [eachItem]
                       [neighborIteration],
                       matriceOfSourceImage
                       [eachItem + 1]
                       [neighborIteration])):
10                           nilaiTreshold += 1
11                     ENDIF
12                  ENDFOR
13                  boxCompareResult.append(
                    [matriceOfSourceImage
                    [eachItem][0],
                    matriceOfSourceImage
                    [eachItem][1],
                    nilaiTreshold])
14              ELSEIF (np.array_equal(
                matriceOfSourceImage[eachItem][2],
                matriceOfSourceImage
                [eachItem - 1][2])
                OR np.array_equal(
                matriceOfSourceImage[eachItem][2],
                matriceOfSourceImage
                [eachItem - 2][2])):
15                  for neighborIteration in
                    range(3,8):
```

```
16                      IF(np.array_equal
                        (matriceOfSourceImage
                        [eachItem]
                        [neighborIteration],
                        matriceOfSourceImage
                        [eachItem - 1]
                        [neighborIteration])):
17                            nilaiTreshold += 1
18                      ENDIF
19                   ENDFOR
20               boxCompareResult.append(
                 [matriceOfSourceImage
                 [eachItem][0],
                 matriceOfSourceImage
                 [eachItem][1], nilaiTreshold])
                 ENDIF
21            ENDIF
22  ENDFUNCTION
```

***Pseudocode* 4.8 Kode Program pengecekan kemiripan antar piksel yang bersebelahan**

Pada *Pseudocode* 4.8 piksel-piksel yang bersebelahan dibandingkan beserta dengan kelima piksel tetangganya. Letak tetangga-tetangga suatu piksel yang akan dibandingkan digambarkan pada Gambar 3.6. Apabila blok tetangga sama persis dengan blok lain yang memiliki urutan *sequence* tetangga seperti pada Gambar 3.6, maka piksel-piksel tersebut ditandai dan diidentifikasi sebagai piksel-piksel yang terkena serangan *copy-move* melalui *Pseudocode* 4.9.

```
1   FUNCTION convertImageValue(self, threshold):
2       thresholdShiftNeighbor <- threshold
3       for x in range
        (0, len( boxCompareResult)):
4           i,j,matchNeighborVal <-
            boxCompareResult[x]
5           IF (matchNeighborVal >=
            thresholdShiftNeighbor):
6                result_image[i][j] <-
                 [255, 255, 255]
7                imageSource[i][j] <-
                 [255, 255, 255]
8           ELSE:
9               break
10          ENDIF
11      ENDFOR
12  ENDFUNCTION
```

***Pseudocode* 4.9 Kode program menandai bagian citra hasil pengecekan *shift neighbour* matching**

4.2.2 Modifikasi metode *exhaustive search*

Hal pertama yang dilakukan pada modifikasi metode *exhaustive* adalah menginisialisasi variabel-variabel yang akan digunakan dalam proses pendeteksian serangan *copy-move* terhadap citra melalui *Pseudocode* 4.10. Inisialisasi variabel yang dilakukan pada tahap ini hampir sama dengan inisialisasi pada metode *exhaustive* sebelumnya.

Variabel-variabel yang akan digunakan antara lain path untuk membaca data masukan dan menyimpan hasil keluaran, array untuk menyimpan piksel hasil konversi, hasil sorting, dan hasil pembentukan tetangga piksel, variabel untuk menyimpan nilai kernel, nilai MSE, dn atribut-atribut yang dibutuhkan pada Tabel 3.1 dalam basis data.

```
1   CLASS class_ExhaustiveModification(object):
        FUNCTION __init__(self):
2           Kernel_X <- 0
3           Kernel_Y <- 0
4           Nilai_Tetangga <- 0
5           MSE <- 0
6           MSE_BLACK <- 0
7           DATE <- 0
8           boxCompareResult <- []
9       ENDFUNCTION
10
11      FUNCTION PathInitialization (self,
        imagePath,pathHasilFile,
        pathKunciJawaban, pathMask):
12          imagePath <- imagePath
13          pathHasilFile <- pathHasilFile
14          pathKunciJawaban <-
            pathKunciJawaban
15          pathMask <- pathMask
            matriceOfSourceImage <- []
16          distinctList <- []
17          canvasForFilteredImage <- []
18      ENDFUNCTION
```

Pseudocode* 4.10 Kode program inisialisasi variabel pada modifikasi metode *exhaustive search

Tahap selanjutnya adalah membaca piksel-piksel pada citra melalui *Pseudocode* 4.11. Pembacaan setiap file dilakukan pada folder yang dituju. Untuk setiap file yang berekstensi PNG akan parsing kedalam kelas pemrosesan citra yang nantinya akan dilakukan pengambilan data bentuk citra yang dituju. Untuk setiap citra akan memiliki nilai ukuran yang akan digunakan untuk membentuk citra buatan yang berwarna hitam yang akan digunakan selama proses pencarian piksel yang terdeteksi sebagai serangan *copy-move.* Informasi yang dimiliki oleh setiap citra dalam bentuk ukuran lebar dan tinggi digunakan juga sebagai parameter perulangan pada proses-proses selanjutnya.

```
1   FUNCTION          LoadTheImageFile          (self,
    pathOriginalImageFile):
2           listGlob <- pathOriginalImageFile
3           fileName <- listGlob
4           fileTemp <- fileName.split('.')
5           fileType <- "." + fileTemp[1]
6           fileName <- fileTemp[0]
7           fileType <-  fileType
8           fileName <-  fileName
9           completeImagePath <-  imagePath +
            fileName + fileType
10          imageSource <-
            cv2.imread(completeImagePath)
11          height, width, channels <-
            imageSource.shape
12          height <- height
13          width <- width
14          channels <- channels
15          OUTPUT "Working on file \t\t\t: " +
            completeImagePath
16  ENDFUNCTION
```

Pseudocode* 4.11 Kode program membaca piksel citra pada modifikasi metode *exhaustive search

Selanjutnya piksel-piksel tersebut dikonversikan ke dalam bentuk *grayscale* dengan mengimplementasikan *library cv2.cvtColor* melalui *Pseudocode* 4.12.

```
1   FUNCTION ConvertRGBToGrayscale(self,):
             grayscaleImage <- cv2.cvtColor(
2            imageSource, cv2.COLOR_RGB2GRAY)
3   ENDFUNCTION
```

***Pseudocode* 4.12 Kode program konversi citra ke grayscale**

Hasil konversi tiap-tiap piksel citra dari RGB ke *grayscale* tersebut diekstrasi menggunakan metode *Gaussian Filter,* atau *Laplacian filtering*. Ekstraksi piksel ini juga disebut tahap *filtering*. Seperti pada metode *exhaustive* sebelumnya, metode *Gaussian Filter* dijalankan melalui *Pseudocode* 4.3. Sedangkan metode *Laplacian Filter* dijalankan melalui *Pseudocode* 4.4.

Tahap selanjutnya adalah melakukan *overlapping block* pada citra dengan ukuran **10x10** melalui *Pseudocode* 4.13.

```
1   FUNCTION InitShiftVector(self):
2            sizeVector <- 10
3            allShiftVector <-
             image.extract_patches_2d(
             grayscaleImage, (sizeVector,
             sizeVector))
4   ENDFUNCTION
```

Pseudocode* 4.13 Kode program overlapping citra pada modifikasi metode *exhaustive search

Hasil *overlapping* citra dengan ukuran **10x10** ini menghasilkan *array* dua dimensi. Sehingga harus dikonversi menjadi *array* satu dimensi terlebih dahulu melalui *Pseudocode* 4.14.

```
1   FUNCTION     IndexConvert1Dto2D(self,     indeksX,
    indeksY, width):
2           RETURN (indeksX * width + indeksY)
3   ENDFUNCTION
```

***Pseudocode* 4.14 Kode program konversi array overlapping 2D menjadi array 1D**

Hasil konversi *array* satu dimensi pada *Pseudocode* 4.13 disimpan ke dalam *array overlapping*. Array ini nantinya akan digunakan pada tahap pengecekan apakah piksel-piksel yang berada dalam satu *block overlapping* memiliki kesamaan dengan *block overlapping* lainnya.

Tahap selanjutnya adalah mengelompokkan piksel-piksel citra yang telah berhasil dibaca melalui *Pseudocode* 4.11. Namun sebelumnya piksel-piksel tersebut dikonversikan terlebih dahulu dari RGB ke *integ*er melalui *Pseudocode* 4.15.

```
1   FUNCTION rgbToInt(self, rgb):
2           RETURN rgb[0] * 65536 + rgb[1] * 256
            + rgb[2]
3   ENDFUNCTION
```

Pseudocode* 4.15 Kode program konversi RGB ke integer pada modifikasi metode *exhaustive search

Hasil konversi setiap piksel citra dari RGB menjadi *integer* tersebut kemudian diekstrasi menggunakan metode *Gaussian* Filter atau Laplacian *Filtering*.

Hasil ekstraksi fitur tiap piksel tersebut dan kelima piksel-piksel tetangganya (sesuai dengan algoritma *neighbour shift matching*) dimasukkan ke dalam sebuah *array* melalui *Pseudocode* 4.5 yang sudah dijelaskan pada metode *exhaustive search* sebelumnya. Selanjutnya *array* tersebut dikonversi ke dalam bentuk *Hash* seperti pada metode *exhaustive search* sebelumnya melalui *Pseudocode* 4.6. Kemudian hasil konversi piksel ke dalam bentuk *Hash* tersebut, diurutkan secara *ascending*. Metode pengelompokan menggunakan *neighbour shifting matching* akan mengelompokan sementara piksel piksel yang memiliki kemiripan secara sekilas atau kemiripan tetangga yang hanya sebanyak 5 piksel.

Tahap selanjutnya adalah mengelompokkan piksel-piksel yang telah diurutkan tersebut berdasarkan kesamaan nilai *Hash* piksel. Proses *sorting* sebelumnya bertujuan untuk memudahkan proses pengelompokkan, karena piksel-piksel yang memiliki kemiripan akan bersebelahan satu dengan yang lainnya. Setelah pengelompokan yang dilakukan pada *sorting*, maka akan dilakukan pengecekan pada setiap piksel dan piksel setelahnya. Untuk setiap piksel yang mirip akan diletakan pada penyimpanan sementara. Ketika piksel yang dibandingkan dengan piksel selanjutnya tidak memiliki kemiripan maka piksel tersebut akan dibandingkan dengan piksel sebelumnya dan penyimpanan sementara tersebut akan di*parsing* kedalam metode yang akan membandingkan kemiripan dengan mengkombinasikan semua

kemungkinan piksel yang ada pada penyimpanan sementara. Pengelompokkan ini dilakukan melalui *Pseudocode* 4.16.

```
1   FUNCTION CheckShiftVector(self):
2         boxCompareResult <- []
3         arrayShiftVector <- []
4         matriceOfSourceImage <-matriceOfSourceImage
5         for eachItem in range
          (0, len(matriceOfSourceImage)):
6               nilaiTreshold <- 0
7               IF (eachItem + 1 <= len(
                matriceOfSourceImage) - 1):
8                     IF(np.array_equal(
                      matriceOfSourceImage
                      [eachItem][8],
                      matriceOfSourceImage
                      [eachItem+1][8])):
9                           arrayShiftVector.append
                            ([matriceOfSourceImage
                            [eachItem][0],
                            matriceOfSourceImage
                            [eachItem][1],
                            matriceOfSourceImage
                            [eachItem][2]])
10                    ELSE:
11                          IF(np.array_equal(
                            matriceOfSourceImage
                            [eachItem-1][8],
                            matriceOfSourceImage
                            [eachItem][8])):
12                                arrayShiftVector.
                                  append([
                                  matriceOfSourceImage
                                  [eachItem][0],
                                  matriceOfSourceImage
                                  [eachItem][1],
                                  matriceOfSourceImage
                                  [eachItem][2])
13                                LocalExhautiveTest
                                  (arrayShiftVector)
14                                arrayShiftVector<- []
```

***Pseudocode* 4.16 Kode Program pengelompokan piksel-piksel yang sama**

Tahap selanjutnya pada masing-masing *cluster* dilakukan pengecekan piksel dengan memanfaatkan *array overlapping block* hasil *overlapping block* sebelumnya. Tiap anggota pada *cluster* dicek dengan anggota lain pada *cluster* yang sama dengan menghitung nilai MSE antara *overlapping block* milik keduanya. Pengecekan hanya dilakukan dengan setiap anggota pada satu *cluster* yang sama dan tidak dilakukan pengecekan dengan piksel-piksel lain yang bukan merupakan anggota piksel dari *cluster* tersebut. Pengecekan *overlapping block* milik piksel-piksel dalam satu *cluster* dilakukan melalui *Pseudocode* 4.17.

```
1  FUNCTION CompareShiftVector(self,arrayOne,
2  arrayTwo):
3      result <- mean_squared_error
       (arrayOne, arrayTwo)
4      RETURN result
5  ENDFUNCTION
```

***Pseudocode* 4.17 Penghitungan nilai MSE antara dua buah overlapping block dalam satu cluster**

Setelah didapatkan hasil MSE dari kedua *overlapping block* pada satu *cluster*, dilakukan pengecekan apakah nilai MSE tersebut lebih dari nilai *threshold* yang telah ditentukan. Nilai *threshold* yang digunakan pada modifikasi metode *exhaustive* ini adalah sebesar **80**. Apabila nilai MSE kedua *overlapping block* pada proses pengecekan bernilai kurang dari **80,** maka kedua *block* tersebut dianggap sebagai piksel-piksel yang terkena serangan *copy-move*. Proses pengecekan nilai MSE ini dilakukan pada *Pseudocode* 4.18.

```
1  MSE_Result <- CompareShiftVector(arrayOne,
   arrayTwo)
2      IF(MSE_Result < 80):
3          boxCompareResult.append
           ([x,y,value])
4          boxCompareResult.append
           ([x2,y2,value])
```

Pseudocode* 4.18 Kode program pengecekan nilai MSE pada modifikasi metode *exhaustive search

Selanjutnya piksel-piksel yang teridentifikasi terkena serangan *copy-move* ditandai dan diberi warna yang berbeda seperti pada metode *exhaustive* sebelumnya yaitu pada *Pseudocode* 4.9.

Sebelum ditandai dan diberi warna melalui *Pseudocode* 4.9, dilakukan pengecekan terlebih dahulu pada piksel-piksel yang memiliki indeks sama dan piksel *Hash* yang sama, agar tidak ada piksel yang ditandai dan dirubah warna dua kali. Pengecekan ini menggunakan fungsi `MakeDataDistinct` yang dijalankan melalui *Pseudocode* 4.19.

```
1   FUNCTION MakeDataDistinct(self):
2       boxCompareResult <- list(set(
        boxCompareResult))
3   ENDFUNCTION
```

***Pseudocode* 4.19 Kode program fungsi `MakeDataDistinct`**

4.2.3 Metode *Autocorrelation*

Proses awal metode *autocorrelation* hampir sama dengan metode *exhaustive search* sebelumnya yaitu menginisialisasi variabel-variabel yang akan digunakan dalam proses pendeteksian serangan *copy-move* terhadap citra melalui *Pseudocode* 4.20. Inisialisasi dilakukan dengan menyiapkan variabel-variabel *array* yang digunakan dalam proses pendeteksian dan *path* mana yang akan digunakan untuk membaca *file-file* citra sebagai data masukan dari sistem. Variabel-variabel *array* tersebut antara lain digunakan untuk membedakan penyimpanan hasil penghitungan *autocorrelation* dengan penyimpanan piksel-piksel asli milik citra yang telah dibaca oleh sistem. Metode *autocorrelation* adalah salah satu metode yang merubah kumpulan piksel menjadi sinyal yang akan dihitung dengan dirinya sendiri. Modifikasi yang dilakukan pada metode *autocorrelation* adalah perubahan ketika melakukan *exhaustive search* yang nantinya akan dirubah menjadi modifikasi metode *exhaustive search.*

```
1   CLASS autocorrelationModification(object):
2       FUNCTION __init__(self):
3             boxCompareResult <- []
4             array_filter <- []
5             array_autocorr <- []
6             array_autocorr1D <- []
7             array_autocorr_result <- []
8             boxAutoCorr <- []
9             boxFilter <- []
10            threshold_autocor <- 3
11            Nama_File <- 0
12            Kernel_X <- 0
13            Kernel_Y <- 0
14            Nilai_Tetangga <- 0
15            MSE <- 0
16            MSE_BLACK <- 0
17            DATE <- 0
18      ENDFUNCTION
19
20      FUNCTION initPath(self,pathNamaFile,
        pathHasilFile, pathKunciJawaban,
        pathMask):
21            pathNamaFile <- pathNamaFile
22            pathHasilFile <- pathHasilFile
23            pathKunciJawaban<- pathKunciJawaban
24            pathMask <- pathMask
25            matriceOfSourceImage <- []
26      ENDFUNCTION
```

Pseudocode* 4.20 Inisialisasi variabel yang digunakan pada modifikasi metode *autocorrelation

Tahap selanjutnya adalah membaca piksel-piksel tiap citra dan dikonversikannya ke dalam bentuk *integer* melalui *Pseudocode* 4.21. Pada metode konversi nilai RGB menjadi nilai *integer* akan mempermudah pencocokan antara piksel-piksel pada citra. Pada citra digital yang digunakan adalah citra digital yang berupa citra digital RGB atau memiliki kombinasi antara warna merah, hijau, dan biru. Bila melakukan pencocokan menggunakan ketiga parameter nilai untuk setiap citra maka akan membutuhkan waktu komputasi yang lebih tinggi karena banyaknya citra akan dikalikan dengan tiga untuk setiap pencocokan. Bila menggunakan

nilai *integer* maka pencocokan setiap piksel hanya melakukan satu kali pencocokan dengan piksel selanjutnya. Parameter yang dimiliki oleh citra pada *dataset* citra yang terkena serangan copy-move akan dirubah parameternya menjadi satu parameter pada Gambar 4.21

```
1   FUNCTION rgbToInt(self, rgb):
2           RETURN rgb[0] * 65536 + rgb[1] * 256 +
    rgb[2]
3   ENDFUNCTION
4
5   FUNCTION           loadTheFileUp           (self,
    pathOriginalImageFile):
6           listGlob <- pathOriginalImageFile
7           fileName <- listGlob
8           fileTemp <- fileName.split('.')
9           fileType <- "." + fileTemp[1]
10          fileName <- fileTemp[0]
11          fileType <-  fileType
12          fileName <-  fileName
13          imageSourceFileFullPath <-
            pathNamaFile+ fileName + fileType
14          imageSource <-
            cv2.imread(imageSourceFileFullPath)
15          height,width,channels <-
            imageSource.shape
16          height <- height
17          width <- width
18          channels <- channels
19          OUTPUT "Working on file \t\t\t: " +
            imageSourceFileFullPath
20  ENDFUNCTION
```

***Pseudocode* 4.21 Kode Program membaca dan konversi piksel citra ke grayscale**

Langkah pertama pada *Pseudocode* 4.21 adalah membaca piksel-piksel citra dengan menggunakan *library cv2* pada *opening*CV. Piksel-piksel citra tersebut disimpan ke dalam sebuah *array* yang selanjutnya dikonversikan ke dalam bentuk *integer* menggunakan persamaan 4.1.

Sama seperti pada metode *exhaustive search*, *noise* hasil dari konversi piksel-piksel tersebut diminimalisasi menggunakan metode *Gaussian Filter* melalui *Pseudocode* 4.3 atau menggunakan metode *Laplacian Filter* melalui *Pseudocode* 4.4.

Implementasi metode *Gaussian Filter* pada *Pseudocode* 4.3 mengimplementasikan *library scipy.ndimage* dengan memasukkan nilai standar deviasi sebesar 0,5. Sedangkan implementasi metode *Laplacian Filter* pada *Pseudocode* 4.4 mengimplementasikan *library scipy.ndimage*. Hasil *Filtering* tiap piksel disimpan terlebih dahulu ke dalam *array filter*.

Selanjutnya hasil *Filtering* tiap piksel tersebut beserta dengan kelima piksel-piksel tetangganya (sesuai dengan algoritma *neighbour shift matching*) dimasukkan ke dalam sebuah *array* melalui *Pseudocode* 4.22.

Pada *Pseudocode* 4.22 hal yang dilakukan adalah mengambil sebuah piksel dan mengidentifikasi kelima piksel-piksel tetangganya. Apabila piksel tetangga tersebut berada pada pojok maupun tepi citra maka piksel–piksel tetangga disebelah kanan, atas, kiri, maupun bawah yang sudah melewati batas ukuran citra nilainya dirubah menjadi 0 karena nilai yang melewati batas ukuran pada citra tidak bisa digunakan karena tidak bisa diambil nilai pada indeks yang keluar dar batasan ukuran citra yaitu lebar dan tinggi. Maka untuk setiap kali inisialisasi variabel akan dicek apakah indeks yang dimaksud ada atau terdapat pada citra dan tidak melewati ukuran (lebar – 1) dan (tinggi – 1) citra. Hal ini dikarenakan indeks perulangan akan dimulai dari indeks 0.

Setelah itu *array* yang berisi indeks sebuah piksel beserta identifikasi mana saja yang termasuk kelima piksel-piksel tetangganya, di-konversi ke dalam bentuk *Hash* melalui *Pseudocode* 4.23. Setelah dibentuk nilai *Hash* yang dibentuk dari setiap nilai piksel pada citra yang diikuti oleh kelima piksel *neighbour shifting*.

```
1   FUNCTION makeAttributeBox(self):
2       height <-  height
3       width <-  width
4       imageSource <-  array_filter
5       for sumbuX in range(0, height):
6           for sumbuY in range(0, width):
7               value <- imageSource[sumbuX, sumbuY]
                IF (sumbuX + 1 <= height - 1):
8                   A <- np.array (imageSource
9                   [sumbuX + 1, sumbuY])
10              ELSE:
11                  A <- 0
12              ENDIF
13              IF (sumbuY + 1 <= width - 1 AND
14              sumbuX + 1 <= height - 1):
                    B <- np.array (imageSource
15                  [sumbuX + 1, sumbuY + 1])
                ELSE:
16                  B <- 0
17              ENDIF
18              IF (sumbuX + 2 <= height - 1):
19                  A2 <- np.array (imageSource
                    [sumbuX + 2, sumbuY])
                ELSE:
20                  A2 <- 0
21              ENDIF
22              IF (sumbuY + 1 <= width - 1 AND
23              sumbuX + 2 <= height - 1 AND
                sumbuX <= height):
                    B2 <- np.array (imageSource
24                  [sumbuX + 2, sumbuY + 1])
                ELSE:
25                  B2 <- 0
26              ENDIF
27              IF (sumbuY + 2 <= width - 1 AND
28              sumbuX + 2 <= height - 1):
                    XX <- np.array (imageSource
29                  [sumbuX + 2, sumbuY + 2])
                ELSE:
30                  XX <- 0
31              ENDIF
32              temporaryNeighbor <-
33              np.hstack (( value, A, A2, B, B2,
                XX))
```

```
                hasil Hash <-  Hash
34              (temporaryNeighbor)

35  matriceOfSourceImage.append([sumbuX,      sumbuY,
36  value, A, A2, B, B2, XX, hasilHash])
37  ENDFUNCTION
38
```

Pseudocode* 4.22 Kode Program pengelompokan atribut piksel tetangga pada metode *autocorrelation

```
1   FUNCTION Hash(self, array):
2       HasilHash <- Hashlib.sha1(array).hexdigest()
3       RETURN hasilHash
4   ENDFUNCTION
```

***Pseudocode* 4.23 Kode program fungsi Hash**

Fungsi *Hash* tersebut digunakan untuk membuat unik nilai dari suatu piksel dan piksel-piksel tetangganya, sehingga meningkatkan akurasi pengecekan kemiripan antar *block* piksel. *block* piksel disini merupakan piksel-piksel tetangga yang digabungkan menjadi satu pada *Pseudocode* 4.13 sebelumnya.

Selanjutnya hasil dari konversi piksel dan piksel-piksel tetangga ke dalam bentuk *Hash* kemudian disimpan ke dalam sebuah *array* citra. Array citra tersebut berisi letak/indeks piksel (sumbu x, sumbu y), nilai piksel dan tetangga piksel, serta nilai hasil *Hash* piksel – piksel tersebut.

Selanjutnya mengelompokkan piksel-piksel yang sama. Pengelompokan piksel pada citra dilakukan dengan mengurutkan piksel-piksel citra yang telah mengalami preprosessing secara *ascending* melalui *Pseudocode* 4.24.

```
1   FUNCTION Sort_ImgVector(self):
2       matriceOfSourceImage.sort
        (key = operator.itemgetter(8))
3   ENDFUNCTION
```

***Pseudocode* 4.24 Kode Program mengurutkan piksel**

Proses pengurutan piksel-piksel citra pada *Pseudocode* 4.24 di atas mengimplementasikan *library sort* pada Python. Sehingga piksel-piksel yang identik akan diletakkan bersebelahan dan berdekatan satu sama lain setelah dirutkan secara ascending. Hasil *sorting* dari *Pseudocode* 4.24 disimpan kembali kedalam *array* citra sebelumnya.

Selanjutnya proses *filtering* pada piksel-piksel citra tadi yang telah menghasilkan *array filter* dihitung nilai *autocorrelation*nya melalui *Pseudocode* 4.25.

Pada *Pseudocode* 4.25 nilai *autocorrelation* tiap piksel citra dihitung menggunakan *library signal* pada *scipy* dengan memanggil fungsi *signal.fftconvolve()*. Fungsi tersebut merupakan fungsi *autocorrelation* yang dilengkapi dengan menggunakan metode *Fast Furiour Transform (*FFT*)* agar lebih cepat dalam proses penghitungannya.

Setelah mendapatkan hasil nilai *autocorrelation* tiap piksel pada citra, selanjutnya hasil tersebut dikelompokkan dengan cara mengurutkan seacara *descending* melalui *Pseudocode* 4.25. Proses pengurutan piksel-piksel citra pada *Pseudocode* 4.25 di bawah mengimplementasikan *library sort* pada Python.

Mengapa diurutkan secara *descending*? Karena konsep dasar dari metode *autocorrelation* ini sendiri dalam pendeteksian *copy-move* pada citra adalah titik puncak pada grafik *autocorrelation* dapat diindikasikan sebagai bagian dari citra yang terkena serangan *copy-move*. Bagian puncak tersebut dapat direpresentasikan sebagai nilai maksimum dari piksel-piksel hasil penghitungan *autocorrelation*.

Sehingga untuk mencari nilai piksel yang memiliki nilai maksimum dari penghitungan *autocorrelation*, terlebih dahulu melakukan pengecekan terhadap indeks hasil *autocorrelation* dengan indeks piksel pada citra awal. Cara pengecekan dilakukan dengan membandingkan satu-satu indeks yang sesuai.

```
1   FUNCTION Autocorr(self):
2       height <-  height
3       width <-  width
4       array_autocorr1D <- np.reshape
        (array_filter, height * width)
5       array_autocorr_result <-
        signal.fftconvolve(array_autocorr1D,
        array_autocorr1D,mode='full')[len(
        array_autocorr1D) - 1:];
6       array_autocorr<-np.reshape(
        array_autocorr_result, (height, width))
7       for sumbuX in range(0, height):
8           for sumbuY in range(0, width):
9                boxAutoCorr.append([sumbuX,
                 sumbuY, array_autocorr[sumbuX]
                 [sumbuY]])
10          ENDFOR
11      ENDFOR
12      boxAutoCorr.sort
13      (key=operator.itemgetter(2))
        boxAutoCorr.sort
14      (key=operator.itemgetter(2),
        reverse=True)
15      boxAutoCorr <- sorted (boxAutoCorr,
        key=lambda s : s[2])
16  ENDFUNCTION
```

Pseudocode* 4.25 Kode program menghitung nilai *autocorrelation

Setelah piksel-piksel hasil penghitungan dari fungsi *autocorrelation* diletakkan berdekatan satu sama lain berdasarkan nilai maksimumnya, selanjutnya mencari piksel pada *array* citra sebelumnya berdasarkan indeks dari nilai maksimum hasil *autocorrelation* yang telah dilakukan pada *Pseudocode* 4.25. Indeks piksel yang memiliki nilai maksimum pada hasil *autocorrelation* dicari satu-satu pada matriks citra awal.

```
1   FUNCTION Autocor_shift(self, mulai, max):
2       max_count <- max
3       IF (max_count <  threshold_autocor):
4           flag <- 0
5           for yy in range (mulai, len(
            matriceOfSourceImage) - 1):
6               IF (flag = 1):
7                   break
8               ELSEIF (
                matriceOfSourceImage[yy][0] =
                boxAutoCorr[max][0] AND
                matriceOfSourceImage[yy][1] =
                boxAutoCorr[max][1]):
9                   flag <- 1
10                  OUTPUT "CHECK SHIFT VECTOR"
11                  CheckShiftVector(yy)
12          max_count <- max_count+1
13          OUTPUT "NEXT AUTOCOR ITER"
14          Autocor_shift(0, max_count)
15  ENDFUNCTION
```

Pseudocode* 4.26 Kode program mencari indeks piksel citra dari nilai maksimum *autocorrelation

Pseudocode 4.26 mencari piksel pada citra yang memiliki nilai *autocorrelation* maksimum. Piksel tersebut beserta piksel tetangganya dibandingkan dengan piksel-piksel tetangga sebelahnya menggunakan metode *neighbour shift matching* yang dijelasakan pada Gambar 3.6 melalui *Pseudocode* 4.27.

Berbeda dengan metode *exhaustive* sebelumnya, saat tahap membandingkan, tidak dimulai dari indeks piksel yang pertama, tapi dari indeks piksel maksimum hasil penghitungan *autocorrelation*. Langkah ini diulangi lagi dengan mencari nilai maksimum selanjutnya, hingga *threshold* yang telah ditentukan tercapai. *Treshold* yang ditentukan dari hasil uji coba adalah 3, sehingga piksel yang akan dibandingkan adalah 3 buah piksel yang memiliki nilai *autocorrelation* maksimum.

```
1   FUNCTION CheckShiftVector(self,index):
2        boxCompareResult <- []
3        matriceOfSourceImage <-
         matriceOfSourceImage
4        for eachItem in range(index,
         len(matriceOfSourceImage)):
5           nilaiTreshold <- 0
6           IF (eachItem + 1 <=
            len(matriceOfSourceImage) - 1):
7               IF (np.array_equal(
                matriceOfSourceImage[eachItem][2],
                matriceOfSourceImage[eachItem+1[2])
                AND
               (matriceOfSourceImage[eachItem][8]=
               matriceOfSourceImage
               [eachItem+1][8])):
8                    for neighborIteration in
                     range(3,8):
9                      IF(np.array_equal
                       (matriceOfSourceImage
                       [eachItem]
                       [neighborIteration],
                       matriceOfSourceImage
                       [eachItem + 1]
                       [neighborIteration])):
10                           nilaiTreshold += 1
11                     ENDIF
12                  ENDFOR
13                  boxCompareResult.append(
                    [matriceOfSourceImage
                     [eachItem][0],
                    matriceOfSourceImage
                    [eachItem][1],
                     nilaiTreshold])
14              ELSEIF (np.array_equal(
                matriceOfSourceImage[eachItem][2],
                matriceOfSourceImage
                [eachItem - 1][2])
                OR np.array_equal(
                matriceOfSourceImage[eachItem][2],
```

```
            matriceOfSourceImage
            [eachItem - 2][2])):
15              for neighborIteration in
                range(3,8):
16                IF(np.array_equal
                  (matriceOfSourceImage
                  [eachItem]
                  [neighborIteration],
                  matriceOfSourceImage
                  [eachItem - 1]
                  [neighborIteration])):
17                      nilaiTreshold += 1
18                ENDIF
19              ENDFOR
20            boxCompareResult.append(
              [matriceOfSourceImage
              [eachItem][0],
              matriceOfSourceImage
              [eachItem][1],
              nilaiTreshold])
21            ENDIF
22        ENDIF
23  ENDFUNCTION
```

Pseudocode* 4.27 Kode Program pengecekan kemiripan dimulai dari piksel maksimum hasil *autocorrelation

Pada *Pseudocode* 4.27 piksel-piksel yang bersebelahan dibandingkan beserta dengan kelima piksel tetangganya sama seperti pada proses metode *exhaustive*, namun indeks yang digunakan bukan dimulai dari 0 tapi dimulai dari indeks piksel maksimum hasil penghitungan metode *autocorrelation*. Letak tetangga-tetangga suatu piksel yang akan dibandingkan digambarkan pada Gambar 3.6. Apabila blok tetangga sama persis dengan blok lain yang memiliki urutan sequence tetangga seperti pada gambar, maka piksel-piksel tersebut ditandai dan diindikasikan sebagai piksel-piksel yang terkena serangan *copy-move* melalui *Pseudocode* 4.9.

4.2.4 Modifikasi Metode *Autocorrelation*

Modifikasi metode *autocorrelation* ini adalah pengembangan dari metode *autocorrelation* sebelumnya. Hal pertama yang dilakukan pada modifikasi metode *autocorrelation* adalah menginisialisasi variabel-variabel yang akan digunakan melalui *Pseudocode* 4.28. Inisialisasi variabel pada modifikasi metode *autocorrelation* ini hampir sama dengan inisialisasi variabel pada metode *autocorrelation* sebelumnya.

```
1   CLASS autocorrelationModification(object):
2       FUNCTION __init__(self):
3           boxCompareResult <- []
4           array_filter <- []
5           array_autocorr <- []
6           array_autocorr1D <- []
7           array_autocorr_result <- []
8           boxAutoCorr <- []
9           boxFilter <- []
10          threshold_autocor <- 3
11          Nama_File <- 0
12          Kernel_X <- 0
13          Kernel_Y <- 0
14          Nilai_Tetangga <- 0
15          MSE <- 0
16          MSE_BLACK <- 0
17          DATE <- 0
18      ENDFUNCTION
19
20      FUNCTION initPath(self,pathNamaFile,
        pathHasilFile, pathKunciJawaban,
        pathMask):
21          pathNamaFile <- pathNamaFile
22          pathHasilFile <- pathHasilFile
23          pathKunciJawaban<- pathKunciJawaban
24          pathMask <- pathMask
25          matriceOfSourceImage <- []
26          distinctList <- []
27      ENDFUNCTION
```

Pseudocode* 4.28 Kode program inisialisasi variabel pada modifikasi metode *autocorrelation

Variabel-variabel yang akan digunakan pada *Pseudocode* 4.28 di atas antara lain path untuk membaca data masukan dan menyimpan hasil keluaran, array untuk menyimpan piksel hasil konversi, hasil *sorting*, hasil pembentukan tetangga piksel, hasil penghitungan nilai autocorrelation, serta variabel-variabel untuk menyimpan nilai kernel, nilai MSE, nilai *treshold* pada penghitungan autocorrelation dan atribut-atribut yang dibutuhkan pada Tabel 3.1 dalam basis data.

Selanjutnya membaca piksel-piksel pada citra transformasikannya ke dalam bentuk *grayscale* dengan mengimplementasikan *library cv2.cvtColor*, seperti yang dilakukan modifikasi metode *exhasutive search* sebelumnya pada *Pseudocode* 4.12. Transformasi dari nilai RGB menjadi nilai *grayscale* akan membuat kombinasi setiap nilai setiap piksel akan menjadi lebih sedikit dibandingkan menggunakan nilai piksel RGB.

Hasil konversi citra RGB ke *grayscale* tersebut selanjutnya diminimalkan *noise*nya menggunakan metode *Gaussian Filter* atau *Laplacian Filter* seperti yang dijelaskan pada masing-masing metode sebelumnya.

Tahap selanjutnya adalah melakukan *overlapping* citra dengan ukuran **10x10,** seperti yang dilakukan pada modifikasi metode *exhaustive* sebelumnya melalui *Pseudocode* 4.13.

Karena hasil *overlapping* citra tersebut adalah *array* dua dimensi, maka harus dikonversi terlebih dahulu menjadi *array overlapping* satu dimensi melalui *Pseudocode* 4.14. Array *overlapping* ini digunakan saat pengecekan nilai MSE antara dua buah *block overlapping*.

Selanjutnya adalah membentuk *array* yang berisi tiap-tiap piksel citra beserta kelima piksel tetangganya, seperti yang dilakukan pada metode *autocorrelation* sebelumnya melalui *Pseudocode* 4.21. Array tersebut diberi nama *array* citra, *array* ini nantinya akan dihitung nilai *autocorrelation*nya. Namun sebelumnya isi piksel-piksel di dalam *array* citra dikonversikan terlebih dahulu ke dalam bentuk *Hash* dan diurutkan secara

ascending seperti yang dilakukan pada metode *autocorrelation* sebelumnya.

Kemudian *array* citra yang telah terurutkan tadi, dihitung nilai Autococorrelationnya pada masing-masing piksel di tiap indeks pada *array* tersebut. Penghitungan *autocorrelation* pada *array* citra ini dilakukan menggunakan *library signal* pada *scipy* dengan memanggil fungsi *signal.fftconvolve()*. Fungsi tersebut merupakan fungsi *autocorrelation* yang dilengkapi dengan implementasi dari metode *Fast Furiour Transform (*FFT*)*, agar lebih cepat dalam proses penghitungannya.

Hasil penghitungan nilai *autocorrelation* tadi diurutkan secara *descending*. Selanjutnya, nilai hasil penghitungan *autocorrelation* yang sudah diurutkan tersebut disimpan kedalam *array autocorrelation*, yang selanjutnya akan dicari nilai maksimumnya dan dikelompokkan.

```
1   FUNCTION Autocor_shift(self, mulai, max):
2       max_count <- max
3       IF (max_count <  threshold_autocor):
4           flag <- 0
5           for yy in range (mulai, len(
            matriceOfSourceImage) - 1):
6               IF (flag = 1):
7                   break
8               ELSEIF (
                matriceOfSourceImage[yy][0] =
                boxAutoCorr[max][0] AND
                matriceOfSourceImage[yy][1] =
                boxAutoCorr[max][1]):
9                   flag <- 1
10                  OUTPUT "CHECK SHIFT VECTOR"
11                  CheckShiftVector(yy)
12          max_count <- max_count+1
13          OUTPUT "NEXT AUTOCOR ITER"
14          Autocor_shift(0, max_count)
15  ENDFUNCTION
```

Pseudocode* 4.29 Kode Program pencarian indeks piksel dari nilai maksimum *autocorrelation

Pengecekan untuk mencari indeks yang memiliki nilai maksimum *autocorrelation* dilakukan melalui *Pseudocode* 4.29. Hasil dari *Pseudocode* 4.29 merupakan indeks suatu piksel pada *array* citra yang telah berbentuk *Hash* dan telah diurutkan secara *ascending*, dimana piksel tersebut memiliki nilai *autocorrelation* maksimum dan dianggap memiliki potensi besar sebagai kandidat piksel yang terkena serangan *copy-move*. Nilai *ascending* yang dihasilkan pada proses ini adalah nilai perhitungan pada proses *autocorrelation* yang akan menghasilkan nilai yang berdasarkan dengan nilai disekitarnya karena nilai *autocorrelation* adalah nilai yang dibandingkan dengan dirinya sendiri, maka nilai yang mempunyai nilai korelas tinggi adalah nilai yang memiliki nilai yang berkorelasi dengan kumpulan nilai sekitarnya.

Tahap selanjutnya adalah mengelompokkan piksel-piksel yang telah diurutkan tersebut berdasarkan kesamaan nilai *Hash* piksel. Proses pengelompokan tidak dimulai dari indeks pertama *array* citra, namun dimulai dari indeks piksel yang memiliki nilai maksimum hasil *autocorrelation* (hasil keluaran pada *Pseudocode* 4.29). Proses *sorting* secara *ascending* pada *array* citra sebelumnya bertujuan untuk memudahkan proses pengelompokkan, karena piksel-piksel yang sama pasti akan bersebelahan. Pengelompokkan ini dilakukan melalui *Pseudocode* 4.30.

Berbeda dengan modifikasi metode *exhaustive* sebelumnya, saat tahap mengelompokkan, tidak dimulai dari indeks piksel yang pertama, tapi dari indeks piksel maksimum hasil penghitungan *autocorrelation* sebelumnya.

Langkah ini diulangi lagi dengan mencari nilai maksimum selanjutnya melalui *Pseudocode* 4.29, hingga *threshold* yang telah ditentukan tercapai. *Treshold* yang ditentukan dari hasil uji coba adalah 3, sehingga indeks piksel yang akan dikelompokkan dimulai dari indeks 3 buah piksel yang memiliki nilai *autocorrelation* maksimum.

Pada *Pseudocode* 4.30 piksel-piksel hasil *Hash array* citra yang bersebelahan dibandingkan kemiripannya sama seperti pada proses modifikasi metode *exhaustive*, namun indeks yang

digunakan bukan dimulai dari 0 tapi dimulai dari indeks piksel maksimum hasil penghitungan metode *autocorrelation*. Indeks piksel yang memiliki nilai maksimum pada hasil *autocorrelation* dicari satu-satu pada matriks citra awal. Selanjutnya Apabila nilai piksel yang bersebelahan telah berbeda, maka piksel disebelahnya tersebut merupakan anggota piksel *cluster* lain.

Tahap selanjutnya pada masing-masing *cluster* dilakukan pengecekan piksel dengan memanfaatkan *array overlapping* hasil *overlapping block* sebelumnya. Tiap anggota pada *cluster* dicek dengan anggota lain pada *cluster* yang sama dengan menghitung nilai MSE antara *overlapping block* yang memiliki indeks yang sama dengan piksel yang dibandingkan dengan piksel pembanding. Pengecekan hanya dilakukan dengan setiap anggota pada satu *cluster* yang sama dan tidak dilakukan pengecekan dengan piksel-piksel lain yang bukan merupakan anggota piksel dari *cluster* tersebut. Pengecekan *overlapping block* milik piksel-piksel dalam satu *cluster* dilakukan melalui *Pseudocode* 4.31. Pengecekan kemiripan antara satu piksel dengan piksel lainnya didasarkan kepada *overlapping block* pada dua nilai piksel yang ditentukan dengan nilai sumbu *x* dan sumbu *y* yang sama dengan kedua piksel yang sedang dibandingkan. Nilai *overlapping block* akan dibandingkan dengan MSE. Nilai MSE antara dua *overlapping block* yang mendekati nilai 0 menunjukan semakin mirip kombinasi/anggota kedua *overlapping block*.

Pengecekan nilai MSE ini dilakukan antara *overlapping block* dalam satu *cluster*, sehingga mempercepat proses pengecekan antara piksel-piksel yang teridentifikasi terkena serangan *copy-move*, dibandingkan dengan mengecek satu-satu antara satu piksel dengan piksel disebelahnya. Karena proses *clustering* sebelumnya telah mengelompokkan piksel-piksel yang sama ke dalam satu *cluster*. Dan proses pengecekan dengan penghitungan nilai MSE dapat meningkatkan akurasi pendeteksian *copy-move* pada citra.

```
    FUNCTION CheckShiftVector(self, indeks):
1        boxCompareResult <- []
2        arrayShiftVector <- []
3        matriceOfSourceImage <-matriceOfSourceImage
4        for eachItem in range (indeks,
5        len (matriceOfSourceImage)):
6           nilaiTreshold <- 0
7           IF(eachItem + 1 <= len
8            (matriceOfSourceImage) - 1):
                IF (np.array_equal
9               (matriceOfSourceImage
                [eachItem][8],
                matriceOfSourceImage[eachItem +
                1][8])):
                     arrayShiftVector.append
10                   ( matricrOfSourceImage
                     [eachItem][0],
                     matriceOfSourceImage
                     [eachItem][1],
                     matriceOfSourceImage
                     [eachItem][2])
                ELSE:
11                   IF (np.array_equal
12                   (matriceOfSourceImage
                     [eachItem-1][8],
                     matriceOfSourceImage
                     [eachItem][8])):
                          arrayShiftVector.
13                        append
                          (matricrOfSourceImage
                          [eachItem][0],
                          matriceOfSourceImage
                          [eachItem][1],
                          matriceOfSourceImage
                          [eachItem][2])
                          LocalExhautiveTest(
14                        arrayShiftVector)
                          arrayShiftVector <- []
15                   END IF
16              END IF
             END IF
         END FOR
    ENDFUNCTION
```

Pseudocode* 4.30 Kode program pengelompokkan piksel pada modifikasi metode *autocorrelation

```
1  FUNCTION        CompareShiftVector(self,arrayOne,
2  arrayTwo):
3      result <- mean_squared_error
       (arrayOne, arrayTwo)
4      RETURN result
5  ENDFUNCTION
```

***Pseudocode* 4.31 Penghitungan nilai MSE antara dua buah overlapping block dalam satu cluster**

Setelah didapatkan hasil MSE dari kedua *overlapping block* pada satu *cluster*, dilakukan pengecekan apakah nilai MSE tersebut lebih dari nilai *threshold* yang telah ditentukan. Nilai *threshold* yang digunakan pada modifikasi metode *exhaustive* adalah sebesar **150**. Apabila nilai MSE kedua *overlapping block* pada proses pengecekan bernilai kurang dari **150,** maka kedua *block* tersebut dianggap sebagai piksel-piksel yang terkena serangan *copy-move*. Proses pengecekan nilai MSE ini dilakukan pada *Pseudocode* 4.32.

```
1  MSE_Result   <-      CompareShiftVector(arrayOne,
   arrayTwo)
2      IF(MSE_Result < 150):
3          boxCompareResult.append
           ([x,y,value])
4          boxCompareResult.append
           ([x2,y2,value])
```

Pseudocode* 4.32 Kode program pengecekan nilai MSE pada modifikasi metode *autocorrelation

Selanjutnya piksel-piksel yang teridentifikasi terkena serangan *copy-move* ditandai dan diberi warna yang berbeda seperti pada metode *exhaustive* sebelumnya yaitu pada *Pseudocode* 4.9.

Sebelum ditandai dan diberi warna melalui *Pseudocode* 4.9, dilakukan pengecekan terlebih dahulu pada piksel-piksel yang

memiliki indeks sama dan piksel *Hash* yang sama, agar tidak ada piksel yang ditandai dan diberi warna dua kali. Pengecekan ini menggunakan fungsi `MakeDataDistinct` yang dijalankan melalui *Pseudocode* 4.19 pada modifikasi metode *exhaustive* sebelumnya. Penandaan hasil yang dilakukan pada proses modifikasi metode *exhaustive search* memungkinkan memasukan piksel yang sama ke dalam penyimpanan kumpulan piksel serta posisi sumbu *x* dan sumbu *y* yang sama karena untuk setiap kali proses pencocokan akan dimasukan kedua nilai yang memenuhi *threshold* kemiripan antara dua piksel. Maka pada penyimpanan tersebut harus dilakukan proses penghapusan nilai piksel dengan sumbu *x* dan sumbu *y* yang sama dengan metode ini. Nilai piksel dengan posisi sumbu yang sama bila tidak dilakukan pembuangan/penghapusan maka akan menghabiskan waktu komputasi yang kurang efektif dan efisien.

4.2.5 *Kernel Morphology*

Proses *morphology* digunakan untuk menghilangkan ketidaksempurnaan bentuk yang ada dalam suatu citra. Dengan operasi *morphology Canny edge detector, closing,* dan *opening*, kombinasi ketiganya dapat menghasilkan citra yang lebih sesuai dengan jawaban yang diinginkan. Proses *morphology* pada citra ini dilakukan melalui *Pseudocode* 4.33. Kombinasi antara *morphology kernel* akan menghasilkan citra yang menunjukan daerah yang terindikasi terkena serangan *copy-move.*

Proses *morphology* pada *Pseudocode* 4.33 menggunakan *library cv2.morphology*. Untuk menentukan nilai *threshold* dari *kernel X* dan *kernel Y* melalui suatu uji coba pada citra, sehingga ditemukan nilai *threshold* terbaik yang dapat digunakan. Setiap kombinasi parameter pada nilai *kernel* akan menghasilkan deteksi serangan *copy-move* yang berbeda.

```
1   FUNCTION filterAndWriteImage
    (self,kernelX_opening,kernelY_opening,kernelX_cl
    osing,kernelY_closing,canny_X,canny_Y):
2       fileName <-  fileName
3       fileType <-  fileType
4       Path_Hasil_File <-  pathHasilFile
5       pathMask <-  pathMask
6       kernelX_opening <- kernelX_opening
7       kernelY_opening <- kernelY_opening
8       kernelX_closing <- kernelX_closing
9       kernelY_closing <- kernelY_closing
10      canny_X <- canny_X
11      canny_Y <- canny_Y
12      hasilGambarCalculasi <-
        Path_Hasil_File + fileName + fileType
13      hasilMaskImage <-
        pathMask + fileName + fileType
14      noFilterImg <-
        Path_Hasil_File + "_UnfilteredA_" +
        fileName +fileType
15      edgedImg <- Path_Hasil_File + fileName
        + "_edged_FilterA" + fileType
16      closingdImg <- Path_Hasil_File + fileName
        + "_closingd_FilterA" + fileType
17      openingedImg <- Path_Hasil_File + fileName
        + "_openinged_FilterA" + fileType
18      cv2.imwrite(hasilMaskImage,
        result_image)
19      cv2.imwrite( noFilterImg,  result_image)
20      image <- cv2.imread( hasilMaskImage)
21      gray <- cv2.cvtColor(image,
        cv2.COLOR_BGR2GRAY)
22      gray <- cv2.GaussianBlur(gray, (3, 3), 0)
23      edged <- cv2.Canny(gray, canny_X,
        canny_Y)
24      kernelClose <- cv2.getStructuringElement
        (cv2.MORPH_RECT,(kernelX_closing,
        kernelY_closing))
25      closingd <- cv2.morphologyEx(edged,
        cv2.MORPH_CLOSE, kernelClose)
        kernelOpen <- np.ones((20, 20),
26      np.uint8)
        openingKernel <- cv2.morphologyEx(closingd,
27      cv2.MORPH_OPEN, kernelOpen)
```

```
     cv2.imwrite( edgedImg,  edged)
28   cv2.imwrite( openingedImg,  openingKernel)
29   cv2.imwrite( closingdImg,  closingd)
30   OUTPUT "Result Image \t\t\t\t: " +
31   hasilGambarCalculasi
     OUTPUT "Result Mask \t\t\t\t: " +
32   hasilMaskImage
     OUTPUT "**********"
33 ENDFUNCTION
34
```

***Pseudocode* 4.33 Kode Program implementasi kernel morfology**

4.2.6 Penghitungan nilai MSE

Penghitungan nilai MSE dilakukan dengan mencocokkan citra hasil keluaran dari sistem dengan citra kunci jawaban. Implementasi penghitungan nilai MSE dilakukan melalui *Pseudocode* 4.34. Nilai MSE pada *Pseudocode* 4.34 menunjukan bahwa ada beberapa kombinasi pencocokan antara citra hasil keluaran perangkat lunak. Nilai MSE yang menjadi acuan penulis adalah nilai MSE yang dihasilkan dari citra hitam yang memiliki ukuran yang sama dengan citra yang terkena serangan *copy-move* dan citra kunci jawaban. Nilai MSE yang dihasilkan antara citra hitam dengan kunci jawaban adalah nilai standar antara citra keluaran setiap proses yang sudah dijalankan. Nilai MSE tersebut adalah nilai citra yang tidak menggunakan *morphology*, nilai citra dengan *morphology opening*, dan *morphology closing* yang dibandingkan dengan kunci jawaban. Nilai masing-masing keluaran perangkat lunak yang mendekati kunci jawaban akan menghasilkan nilai MSE yang lebih kecil dibandingkan antara nilai MSE citra hitam dan citra kunci jawaban. Nilai MSE akan menunjukan kedekatan antara kedua matriks atau *array* yang mempunyai ukuran yang sama besarnya. Semakin kecil nilai MSE maka menunjukan kemiripan antara kedua objek *array* atau matriks yang dibandingkan.

```
1   FUNCTION valueOfMSE(self):
        fileNameKeyConvert<-
        fileName.replace("_copy_", "_gt_")
2       imageKeyFullPath <-  pathKunciJawaban +
        fileNameKeyConvert +  fileType
3       unfilteredImage <- cv2.imread(
        hasilMaskImage)
4       kunciJawaban <-
        cv2.imread(imageKeyFullPath)
5       valuePrediction <- cv2.imread(
        openingedImg)
6       valueCloseImage <- cv2.imread(
        closingdImg)
7       valueKey <- np.array( kunciJawaban)
8       valueKey <-  valueKey.flatten()
9       valuePrediction <- np.array(
        valuePrediction)
10      valuePrediction <-
        valuePrediction.flatten()
11      valueCloseImage <- np.array(
        valueCloseImage)
12      valueCloseImage <-
        valueCloseImage.flatten()
13      blackMask <- np.array( blackMask)
        blackMask <-  blackMask.flatten()
14      unfilteredImage <- np.array(
        unfilteredImage)
15      unfilteredImage <-
        unfilteredImage.flatten()
16      MSE_OPEN <- mean_squared_error(
        valueKey,  valuePrediction)
17      MSE_CLOSE <- mean_squared_error(
        valueKey,  valueCloseImage)
18      MSE_BLACK <- mean_squared_error(
        valueKey,  blackMask)
19      MSE_UNFILTERED <- mean_squared_error(
        valueKey,  unfilteredImage)
20      OUTPUT "Hasil MSE  Unfiltere image\t: "
        + str( MSE_UNFILTERED)
        OUTPUT "Hasil MSE  opening\t\t\t\t: " + str(
21      MSE_OPEN)
        OUTPUT "Hasil MSE  closing\t\t\t: " + str(
22      MSE_CLOSE)
        OUTPUT "Hasil MSE Black Mask \t\t: " +
```

```
23      str( MSE_BLACK)
    ENDFUNCTION
24
```

***Pseudocode* 4.34 Kode Program penghitungan nilai MSE**

4.2.7 Penyimpanan basis data SQLite

Proses penyimpanan ke basis data SQLite dilakukan setelah algoritma yang digunakan (*exhaustive search*, modifikasi metode *exhaustive search*, *autocorrelation* dan modifikasi metode *autocorrelation*) selesai dan menghasilkan nilai yang ingin disimpan seperti nilai *mean squared error* (MSE) antara citra digital hasil dan citra digital kunci jawaban. Penyimpanan kedalam basis data SQLite akan mempermudah penulis menganalis dan mendapatkan hasil dari setiap keluaran citra dari proses perangkat lunak yang langsung dapat dibandingkan dengan citra kunci jawaban. Selain menyimpan hasil perbandingan antara citra kunci jawaban dan citra hasil keluaran perangkat lunak pendeteksi serangan *copy-move,* nilai pada basis data yang ada adalah nilai parameter untuk setiap *morphology kernel* yang digunakan. Nilai pada setiap parameter *morphology kernel* akan merubah jalannya proses perangkat lunak dalam menemukan serangan *copy-move*. Hasil yang juga disimpan pada basis data adalah nilai waktu yang dibutuhkan untuk setiap kali satu citra diproses menjadi beberapa citra yang menunjukan posisi serangan citra. Sebelum melakukan eksekusi pada basis data, proses yang dilakukan adalah melakukan koneksi dengan basis data, bila koneksi yang dicoba gagal maka perangkat lunak akan memberikan pesan *error* dan bila tidak ada kesalahan atau pesan *error* maka hasil-hasil dari metode-metode pemrosesan data akan disusun kedalam bahas SQL. Setelah penyusunan hasil-hasil kedalam bahasa SQL maka akan dicoba untuk melakukan *insert* bila ada pesan *error* maka proses akan berhenti dan penulis akan mendapatkan pesan *error* tersebut, bila tidak ada pesan *error* yang muncul maka nilai-nilai yang sudah disusun akan tersimpan pada basis data yang telah ditentukan.

```
1   class Database:

2       def__init__(self,arrayHasil,
        timeConsumed, statusautocorrelation):
3           self.arrayHasil = arrayHasil
4           self.timeConsumed = timeConsumed
5           self.statusautocorrelation =
            statusautocorrelation
6       def Connection(self):
7           try:
8               self.conn =
                sqlite3.connect
                ('../Database_only/tugasAkhir.db')
9           except Exception as e:
10              import logging
                logging.error
                (traceback.format_exc())
11      def Execute(self):
12          try:
13              self.conn.execute(
                "INSERT INTO Hasil (Nama_File,
                Kernel_X_opening,Kernel_Y_opening,
                Kernel_X_closing,Kernel_Y_closing,
                Canny_X,Canny_Y, Nilai_Tetangga,
                MSE_Unfiltered,MSE_opening,
                MSE_closing,MSE_BLACK,DATE,
                Autocorr_Status,
                Time_Calculation) VALUES
               (\""+str(self.arrayHasil[0])+
               "\",\""+str(self.arrayHasil[1]) +
               "\",\"" + str(self.arrayHasil[2])
               +"\",\""+ str(self.arrayHasil[3])
               +"\",\""+ str(self.arrayHasil[4])
               +"\",\""+ str(self.arrayHasil[5])
               +"\",\""+ str(self.arrayHasil[6])
               +"\",\""+ str(self.arrayHasil[7])
               +"\",\""+ str(self.arrayHasil[8])
               +"\",\""+str(self.arrayHasil[9])
               +"\",\""+str(
               self.arrayHasil[10])
                +"\",\""+ str
                (self.arrayHasil[11])
                +"\",\""  + str
                (self.arrayHasil[12])+"\",\"" +
```

```
            str(self.statusautocorrelation)
            +"\",\"" + str
14          (self.timeConsumed)+ "\")")
15      except Exception as e:
16          import logging
            logging.error
17          (traceback.format_exc())
18  def Commit(self):
19      try:
20          self.conn.commit()
21      except Exception as e:
22          import logging
23          logging.error
            (traceback.format_exc())
24  def closing(self):
25      try:
26          self.conn.closing()
27      except Exception as e:
28          import logging
            logging.error
            (traceback.format_exc())
```

***Pseudocode* 4.35 Kode Program penyimpanan ke basis data SQLite**

(Halaman ini sengaja dikosongkan)

BAB V
HASIL UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai skenario uji coba dan evaluasi pada deteksi serangan *copy-move* pada *dataset* citra yang terkena serang *copy-move*. Hasil uji coba didapatkan dari implementasi pada bab 4 dengan skenario yang berbeda. Bab ini berisikan pembahasan mengenai lingkungan pengujian, data pengujian, dan uji kinerja.

5.1 Lingkungan Pengujian

Lingkungan pengujian pada uji coba permasalahan pendeteksian serangan *copy-move* pada citra dengan metode *exhaustive search* dan *autocorrelation* menggunakan spesifikasi keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Lingkungan Pengujian

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel(R) Core(TM) i5-2.6 GHz
	Memori	10 GB 1600 MHz DDR3
Perangkat Lunak	Sistem Operasi	Windows 8 dan Ubuntu 14.04
	Perangkat Pengembang	PyCharm Ultimate dan DB Browser for SQLite

5.2 Data Pengujian

Subbab ini menjelaskan mengenai data yang digunakan pada uji coba. Seperti yang telah dijelaskan sebelumnya, data terdiri dari 35 buah citra yang mengalami serangan *copy-move* dan 35 buah

citra kunci jawaban yang menunjukan bagian-bagian yang dideteksi terkena serangan *copy-move*.

Data tersebut kemudian diolah menggunakan *Gaussian* dan *Laplacian Filter* sehingga menghasilkan piksel – piksel matriks yang telah diminimalisasikan jumlah *noise* dari sebelumnya. Data hasil preprossesing tersebut akan dikelompokkan pada proses klasifikasi dan *sorting*. Selanjutnya dilakukan pengecekan terhadap piksel-piksel tetangga menggunakan algoritma *neighbour shift matching*.

5.3 Preprocessing citra

Preprocessing citra yang digunakan dalam skenario uji coba adalah pada tahap *filtering* yaitu tahap untuk meminimalisasikan *noise* pada suatu citra. Metode *filtering* yang dibandingkan dalam skenario uji coba ini adalah metode *Gaussian*, *Gaussian Blur*, dan *Laplacian Filter* yang telah dijelaskan pada bab sebelumnya.

5.4 Skenario Uji Coba

Sebelum melakukan uji coba, perlu ditentukan skenario yang akan digunakan dalam uji coba. Melalui skenario ini, perangkat akan diuji apakah sudah berjalan dengan benar dan bagaimana performa pada masing-masing skenario. Dan membandingkan skenario manakah yang memiliki hasil lebih baik. Terdapat 10 macam skenario uji coba, yaitu:

1. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *exhaustive* dan *Gaussian Filter*
2. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *exhaustive* dan *Laplacian Filter*
3. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *exhaustive* dan tanpa menggunakan metode *Filter*

4. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *autocorrelation* dan *Laplacian Filter*
5. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *autocorrelation* dan *Gaussian Filter*
6. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan modifikasi metode *exhaustive* tanpa metode *Filter*
7. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan modifikasi metode *exhaustive* dan *Gaussian Filter*
8. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan modifikasi metode *exhaustive* dan *Laplacian Filter*
9. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode modifikasi *autocorrelation* dan *Gaussian Filter*
10. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode modifikasi *autocorrelation* dan *Laplacian Filter*
11. Penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *naive exhaustive*

5.4.1 Skenario Uji Coba 1

Skenario uji coba 1 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *exhaustive* dan *preprocessing* dengan *Gaussian Filter*. Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *morphology kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSE kurang dari nilai MSE *black* citra yang

digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 1 dapat dilihat pada Tabel 5.2.

Tabel 5.2 Akurasi metode *exhaustive* dan *Gaussian Filter*

Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	31	4	**88,57%**
Kernel opening	0	35	0%
Kernel closing	3	32	8,57%

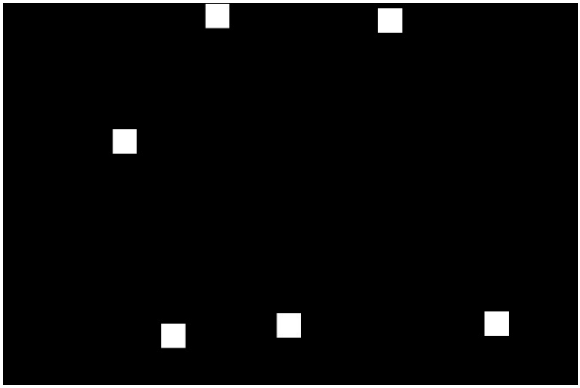
Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, metode *exhaustive* dengan *preprocessing Gaussian Filter* memiliki akurasi tinggi apabila tidak menggunakan *kernel* yaitu mencapai **88,57%.** Sedangkan apabila menggunakan *kernel opening* maupun *kernel closing* akurasi diperoleh sangat rendah yaitu mendekati 0%.

Hasil keluaran metode *exhaustive* dengan *Gaussian Filter* yang menggunakan *kernel opening* dan *kernel closing* sangat menjauhi dari hasil yang diharapkan. Hal ini dikarenakan apabila penggunaan *morphology kernel* diimplementasikan pada hasil keluaran yang sudah baik dari metode *exhaustive*, maka hasil keluaran dari *morphology kernel* menjadi tidak optimal. Ketidakoptimalan hasil keluaran dari *morphology kernel* tersebut dikarenakan kernel *close* tidak menandai bagian/area dari hasil keluaran metode exhaustive, karena sudah dianggap sesuai dengan hasil yang diharapkan. Fungsi utama dari morphology kernel adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang sudah baik, penggunaan *morphology kernel* menjadi tidak optimal.

Contoh citra data masukan pada uji coba satu ada pada Gambar 5.1. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.2. Seperti yang dijelaskan sebelumnya setiap citra data masukan memiliki pasangan kunci jawabannya masing-masing.

Kunci jawaban tersebut berisi citra yang sudah ditandai dan dipastikan benar pada bagian dari citra yang terkena serangan *copy-move*.

Gambar 5.1 Contoh citra masukan pada uji coba 1

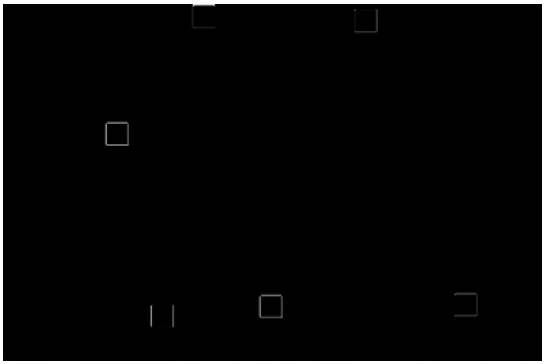


Gambar 5.2 Citra kunci jawaban pada uji coba 1

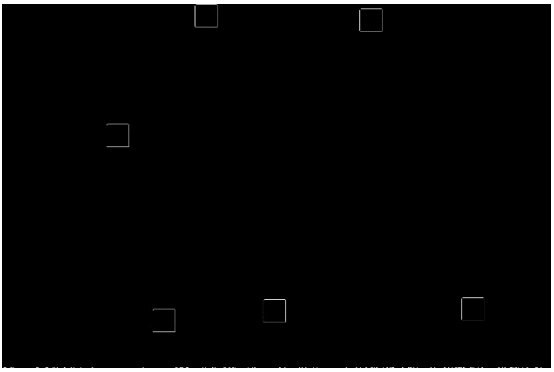
Hasil uji coba 1 pada citra data masukan pada Gambar 5.1 di atas menggunakan *kernel opening* ada pada Gambar 5.3, menggunakan *kernel closing* ada pada Gambar 5.4, menggunakan *kernel edge* ada pada Gambar 5.5, dan tanpa menggunakan *kernel* ada pada Gambar 5.6. Hasil uji coba yang dilakukan adalah hasil uji coba dari keluaran perangkat lunak pendeteksi *copy-move.*

Citra keluaran dengan *morphology opening* adalah citra yang mendapatkan *preprocessing* sebelumnya menggunakan beberapa *morphology kernel* seperti *morphology closing*, *egde*, dan *Canny*. Pengujian dilakukan dengan membandingkan setiap citra keluaran perangkat lunak pendeteksi serangan *copy-move* dengan citra kunci jawaban. Nilai perbandingan antara setiap citra keluaran dan citra hitam adalah pembanding awal yang dapat digunakan untuk menentukan akurasi citra keluaran. Bila nilai citra keluaran yang dibandingkan dengan kunci jawaban lebih kecil dibadingkan nilai citra hitam dan nilai citra kunci jawaban, maka dilakukan pengecekan langsung menggunakan metode *self assesment.*

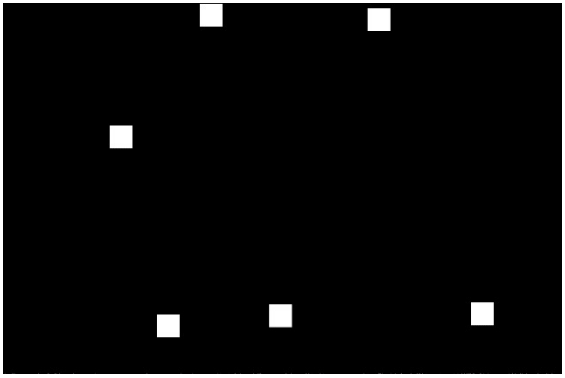
Gambar 5.3 Hasil uji coba 1 menggunakan *kernel opening*



Gambar 5.4 Hasil uji coba 1 menggunakan *kernel closing*

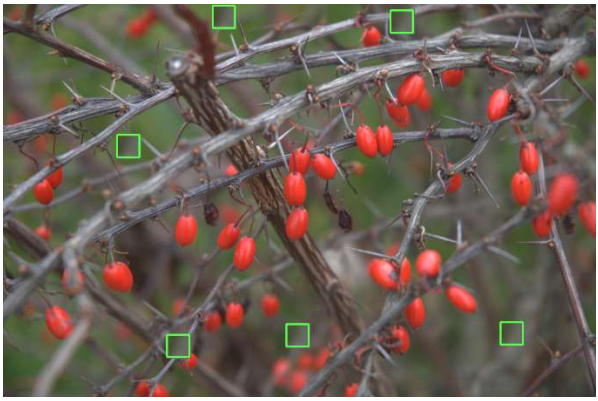


Gambar 5.5 Hasil uji coba 1 menggunakan *kernel edge*



Gambar 5.6 Hasil uji coba 1 tanpa menggunakan *morphology kernel*

Hasil akhir dari uji coba pertama ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan metode *exhaustive search* dan *Gaussian Filter.* Citra hasil keluaran pada uji coba ke 1 ada pada Gambar 5.7.



Gambar 5.7 Hasil akhir uji coba 1

5.4.2 Skenario Uji Coba 2

Skenario uji coba 2 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *exhaustive* dan preprocessing dengan *Laplacian Filter*. Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset yang dikalikan 100%. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSE kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 2 dapat dilihat pada Tabel 5.3.

Tabel 5.3 Akurasi metode *exhaustive* dan *Laplacian Filter*

Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	30	5	**85,71%**
Kernel opening	0	35	0%
Kernel closing	0	35	0%

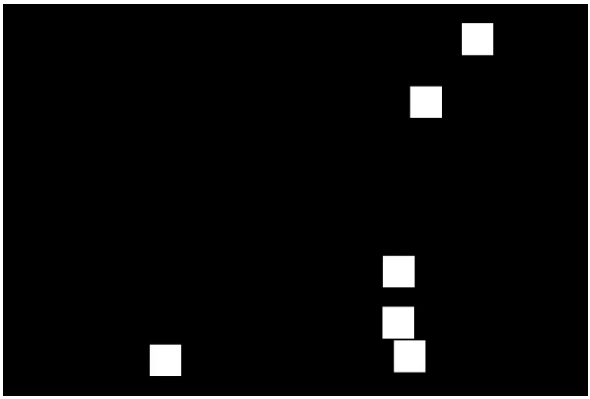
Berdasarkan hasil yang ditunjukkan pada perhitungan akurasi diatas, metode *exhaustive* dengan *preprocessing Laplacian Filter* memiliki akurasi tinggi apabila tidak menggunakan *kernel* yaitu mencapai **85,71%.** Sedangkan apabila menggunakan *kernel opening* maupun *kernel closing* akurasi sangat rendah yaitu mencapai 0%.

Hasil keluaran metode *exhaustive* dengan *Laplacian Filter* yang menggunakan *kernel opening* dan *kernel closing* sangat menjauhi dari hasil yang diharapkan. Hal ini dikarenakan apabila penggunaan *morphology kernel* diimplementasikan pada hasil keluaran yang sudah baik dari metode *exhaustive*, maka hasil keluaran dari *morphology kernel* menjadi tidak optimal. Ketidakoptimalan hasil keluaran dari *morphology kernel* tersebut dikarenakan kernel *close* tidak menandai bagian/area dari hasil keluaran metode exhaustive, karena sudah dianggap sesuai dengan hasil yang diharapkan. *Kernel opening* dibentuk dari hasil keluaran *kernel closing*, sehingga apabila *kernel closing* tidak dapat menghasilkan suatu hasil yang optimal, maka *kernel opening* pun menghasilkan keluaran yang jauh dari hasil yang diharapkan.

Fungsi utama dari morphology kernel adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang sudah baik, penggunaan *morphology kernel* menjadi tidak optimal.

Contoh citra data masukan pada uji coba dua ada pada Gambar 5.8. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.9. Seperti yang dijelaskan sebelumnya setiap citra data masukan memiliki pasangan kunci jawabannya masing-masing. Kunci jawaban tersebut berisi citra yang sudah ditandai dan dipastikan benar pada bagian dari citra yang terkena serangan *copy-move*.

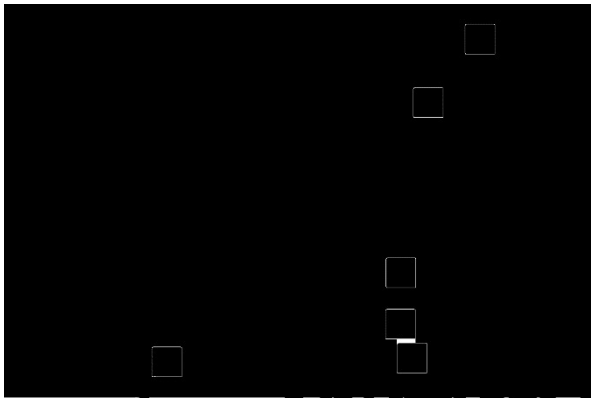
Gambar 5.8 Contoh citra masukan pada uji coba 2



Gambar 5.9 Citra kunci jawaban pada uji coba 2

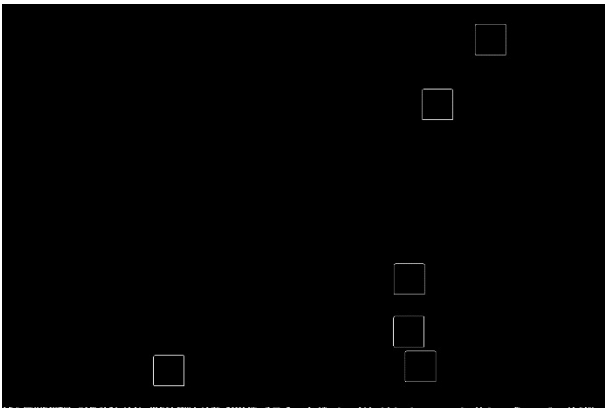
Hasil uji coba 2 pada citra data masukan pada Gambar 5.8 menggunakan *kernel opening* ada pada Gambar 5.10, menggunakan *kernel closing* ada pada Gambar 5.11, menggunakan *kernel edge* ada pada Gambar 5.12, dan tanpa menggunakan *kernel* ada pada Gambar 5.13.

Gambar 5.10 Hasil uji coba 2 menggunakan *kernel opening*

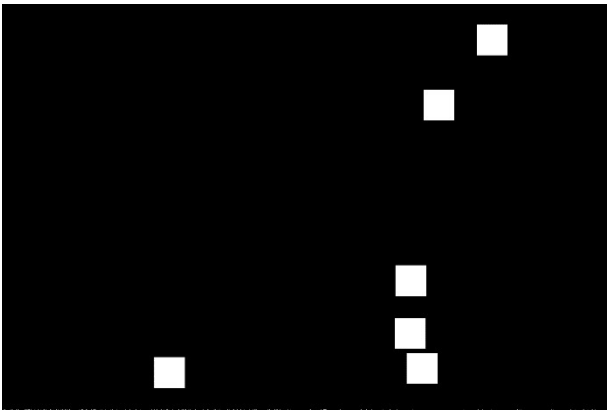


Gambar 5.11 Hasil uji coba 2 menggunakan *kernel closing*

Citra hasil keluaran menggunakan *kernel closing* pada Gambar 5.11 memperlihatkan adanya bagian pada citra yang terlihat putih karena dianggap memenuhi kriteria *kernel closing* yang disebabkan oleh adanya dua buah objek *kernel* edge pada Gambar 5.12 yang berdekatan.

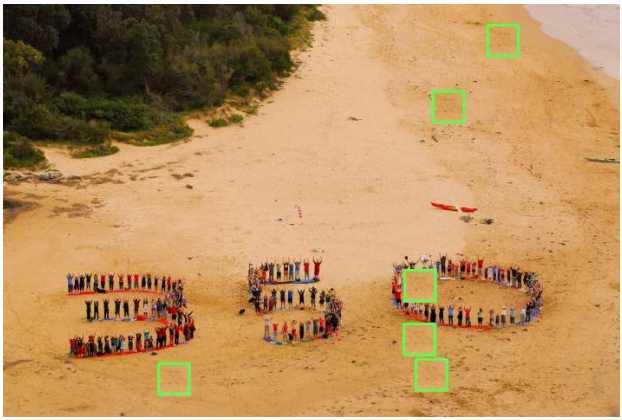


Gambar 5.12 Hasil uji coba 2 menggunakan *kernel edge*



Gambar 5.13 Hasil uji coba 2 tanpa menggunakan *morphology kernel*

Hasil akhir dari uji coba pertama ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan metode *exhaustive search* dan *Laplacian Filter.* Citra hasil keluaran pada uji coba ke 1 ada pada Gambar 5.14.



Gambar 5.14 Hasil akhir uji coba 2

5.4.3 Skenario Uji Coba 3

Skenario uji coba 3 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *exhaustive* dan tanpa menggunakan metode *filtering* pada tahap *preprocessing*. Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSE kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 3 dapat dilihat pada tabel 5.4.

Tabel 5.4 Akurasi metode *exhaustive* dan tanpa *Filter*

Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	24	11	**68,57%**
Kernel opening	0	35	0%
Kernel closing	1	34	2,86%

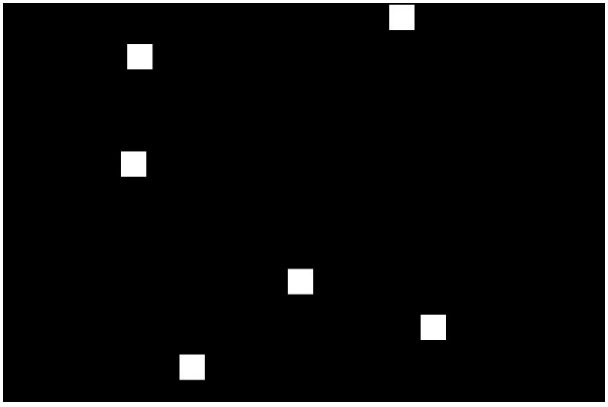
Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, metode *exhaustive* dan tanpa menggunakan metode *Filter* pada tahap *preprocessing*, memiliki akurasi tinggi apabila tidak menggunakan *kernel* yaitu mencapai **68,57%.** Sedangkan apabila menggunakan *kernel opening* maupun *kernel closing* akurasi sangat rendah yaitu mendekati 0%.

Hasil keluaran metode *exhaustive* tanpa metode *Filter* yang menggunakan *kernel opening* dan *kernel closing* sangat menjauhi dari hasil yang diharapkan. Hal ini dikarenakan apabila penggunaan *morphology kernel* diimplementasikan pada hasil keluaran yang sudah baik dari metode *exhaustive*, maka hasil keluaran dari *morphology kernel* menjadi tidak optimal. Ketidakoptimalan hasil keluaran dari *morphology kernel* tersebut dikarenakan kernel *close* tidak menandai bagian/area dari hasil keluaran metode exhaustive, karena sudah dianggap sesuai dengan hasil yang diharapkan. *Kernel opening* dibentuk dari hasil keluaran *kernel closing*, sehingga apabila *kernel closing* tidak dapat menghasilkan suatu hasil yang optimal, maka *kernel opening* pun menghasilkan keluaran yang jauh dari hasil yang diharapkan.

Fungsi utama dari *morphology kernel* adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang sudah baik, penggunaan *morphology kernel* menjadi tidak optimal.

Contoh citra data masukan pada uji coba tiga ada pada Gambar 5.15. sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.16. Seperti yang dijelaskan sebelumnya setiap citra data masukan memiliki pasangan kunci jawabannya masing-masing. Kunci jawaban tersebut berisi citra yang sudah ditandai dan dipastikan benar pada bagian dari citra yang terkena serangan *copy-move.*

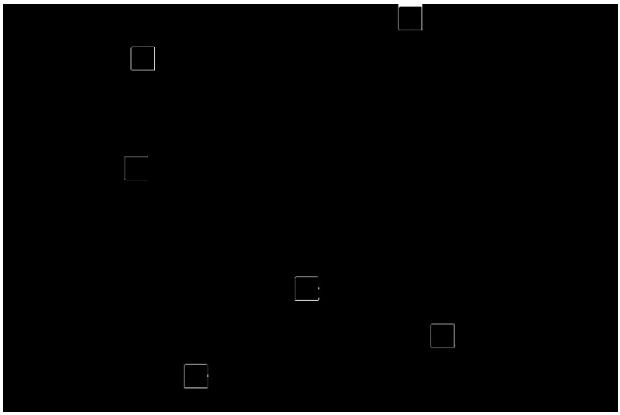
Gambar 5.15 Contoh citra masukan pada uji coba 3



Gambar 5.16 Citra kunci jawaban pada uji coba 3

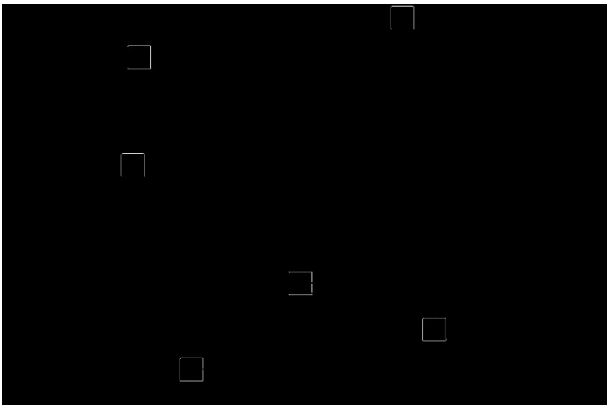
Hasil uji coba 3 pada citra data masukan pada Gambar 5.15 di atas menggunakan *kernel opening* ada pada Gambar 5.17, menggunakan *kernel closing* ada pada Gambar 5.18, menggunakan *kernel edge* ada pada Gambar 5.19, dan tanpa menggunakan *kernel* ada pada Gambar 5.20.

Gambar 5.17 Hasil uji coba 3 menggunakan *kernel opening*

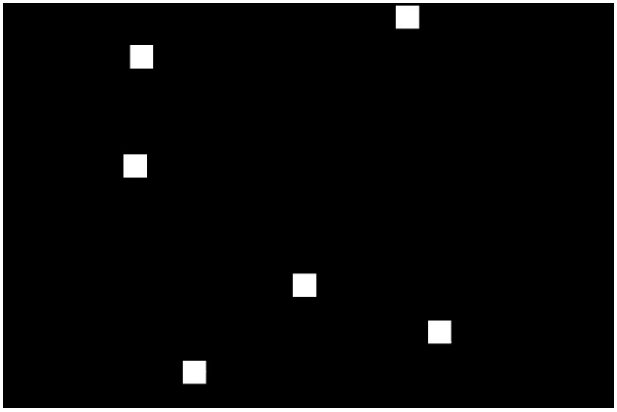


Gambar 5.18 Hasil uji coba 3 menggunakan *kernel closing*

Citra hasil keluaran menggunakan *kernel closing* pada Gambar 5.11 memperlihatkan adanya bagian pada citra yang terlihat putih karena dianggap memenuhi kriteria *kernel closing* yang disebabkan oleh adanya dua buah objek *kernel* edge pada Gambar 5.12 yang berdekatan.



Gambar 5.19 Hasil uji coba 3 menggunakan *kernel edge*



Gambar 5.20 Hasil uji coba 3 tanpa menggunakan *morphology kernel*

Hasil akhir dari uji coba pertama ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan metode *exhaustive search* tanpa metode *Filter.* Citra hasil keluaran pada uji coba ke 1 ada pada Gambar 5.21.

Gambar 5.21 Hasil akhir uji coba 3

5.4.4 Skenario Uji Coba 4

Skenario uji coba 4 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *autocorrelation* dan preprocessing dengan *Laplacian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSE kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 4 dapat dilihat pada Tabel 5.5.

Tabel 5.5 Akurasi metode *autocorrelation* dan *Laplacian Filter*

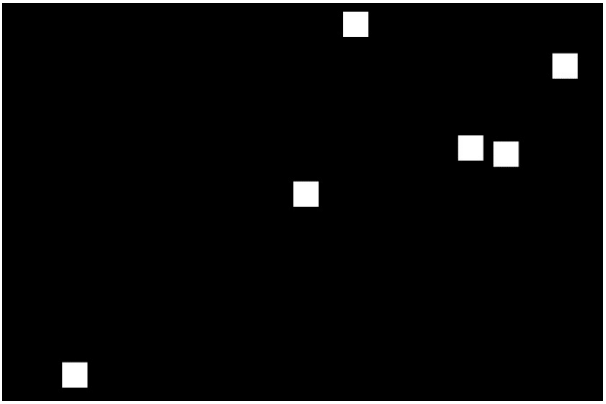
Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	18	17	51,43%
Kernel opening	22	13	**62,86%**
Kernel closing	21	14	60%

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, *autocorrelation* dengan *Laplacian Filter*, memiliki akurasi tinggi apabila menggunakan *kernel opening* yaitu mencapai **62,86%.** Sedangkan apabila menggunakan *kernel closing* hasil yang diperoleh juga hampir sama dengan *kernel opening* yaitu mencapai 60%. Akurasi mencapai nilai terendah apabila tidak menggunakan *kernel* yaitu 51,43%. Pada percobaaan uji skenario menggunakan algoritma *autocorrelation* yang diberikan dua buah kombinasi *filter* antara *Laplacian Filter* pada pengolahan awal citra dan *Gaussian Blur* pada citra yang sudah diberikan tanda letak-letak piksel yang terkena serangan *copy-move.* Hasil pengolahan citra menggunakan *Laplacian Filter* yang kurang baik dapat ditingkatkan menggunakan *morphology kernel*. Hasil menggunakan *morphology kernel* dapat mengurangi jumlah *noise* yang muncul pada citra yang belum diberikan *morphology kernel.* Oleh karena itu hasil keluaran deteksi copy-move menggunakan *autocorrelation* dengan *Laplacian Filter* menjadi lebih tinggi akurasinya apabila menggunakan *morphology kernel.*

Fungsi utama dari *morphology kernel* adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang belum baik, penggunaan *morphology kernel* menjadi lebih optimal dan dapat meningkatkan akurasi pendeteksian.

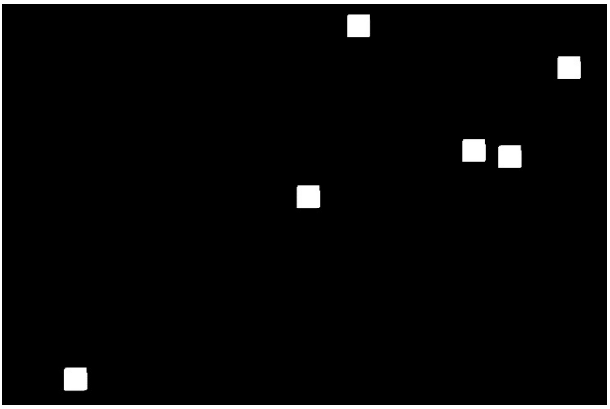
Contoh citra data masukan pada uji coba empat ada pada Gambar 5.22. sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.23. Seperti yang dijelaskan sebelumnya setiap citra data masukan memiliki pasangan kunci jawabannya masing-masing. Kunci jawaban tersebut berisi citra yang sudah ditandai dan dipastikan benar pada bagian dari citra yang terkena serangan *copy-move.*

Gambar 5.22 Contoh citra masukan pada uji coba 4

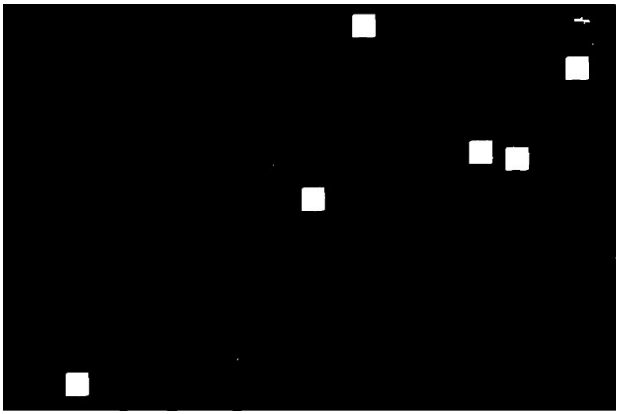


Gambar 5.23 Citra kunci jawaban pada uji coba 4

Hasil uji coba 4 pada citra data masukan pada Gambar 5.22 di atas menggunakan *kernel opening* ada pada Gambar 5.24, menggunakan *kernel closing* ada pada Gambar 5.25, menggunakan *kernel edge* ada pada Gambar 5.26, dan tanpa menggunakan *kernel* ada pada Gambar 5.27.

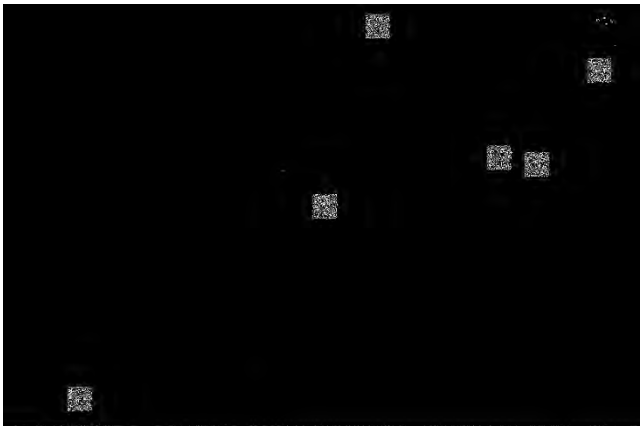


Gambar 5.24 Hasil uji coba 4 menggunakan *kernel opening*

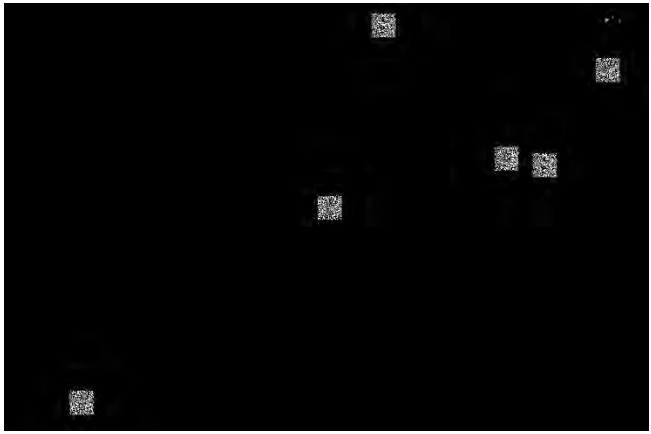


Gambar 5.25 Hasil uji coba 4 menggunakan *kernel closing*

Dapat terlihat perubahan yang terdapat pada Gambar 5.25 kemudian Gambar 5.24. Perubahan yang dapat dilihat adalah adanya penghapusan *noise* yang muncul menggunakan *kernel opening.* Bila suatu objek tidak memenuhi parameter pada *kernel opening*, maka objek/kumpulan piksel tersebut akan diwarnai/dikonversi menjadi warna *background*/hitam.



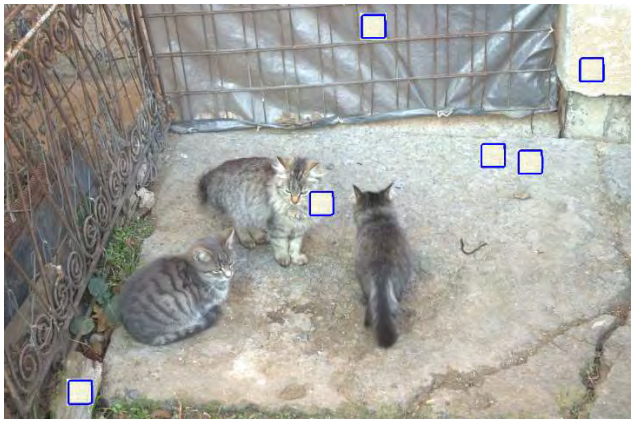
Gambar 5.26 Hasil uji coba 4 menggunakan *kernel edge*



Gambar 5.27 Hasil uji coba 4 tanpa menggunakan *morphology kernel*

Hasil akhir dari uji coba pertama ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa

kernel untuk mendeteksi *copy-move* menggunakan metode *autocorrelation* dan *Laplacian Filter.* Citra hasil keluaran pada uji coba ke 4 ada pada Gambar 5.28.



Gambar 5.28 Hasil akhir uji coba 4

5.4.5 Skenario Uji Coba 5

Skenario uji coba 5 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *autocorrelation* dan preprocessing dengan *Gaussian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSE kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 5 dapat dilihat pada Tabel 5.6.

Tabel 5.6 Akurasi metode *autocorrelation* dan *Gaussian Filter*

Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	25	10	71,43%

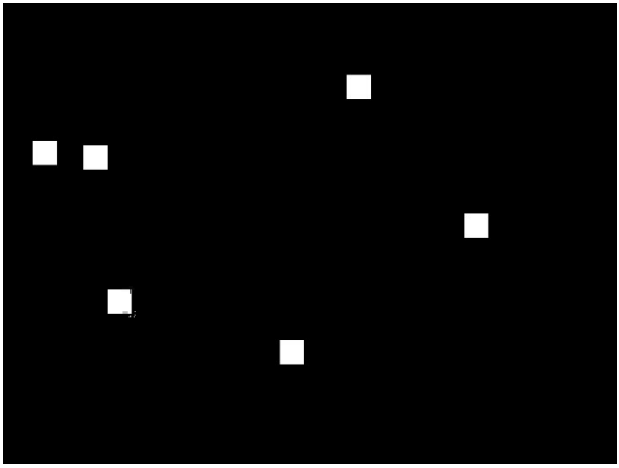
Kernel opening	31	4	**88,57%**
Kernel closing	30	5	85,71%

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, *autocorrelation* dengan *Gaussian Filter*, memiliki akurasi tinggi apabila menggunakan *kernel opening* yaitu mencapai **88,57%.** Sedangkan apabila menggunakan *kernel closing* hasil yang diperoleh juga hampir sama dengan *kernel opening* yaitu mencapai 85,71%. Akurasi mencapai nilai terendah apabila tidak menggunakan *kernel* yaitu 71,43%. Dapat diketahui bahwa citra *morphology kernel opening* yang diperoleh dari *morphology kernel closing* menghasilkan nilai yang lebih baik karena dapat mengurangi jumlah *closing* yang tidak diinginkan. Kecenderungan *closing* adalah tidak membentuk kumpulan piksel yang besarnya tidak memenuhi kriteria parameter *kernel opening,* sehingga dapat dengan mudah *noise-noise* yang terdapat pada citra dapat dirubah atau dikonversi menjadi piksel hitam/*background.*

Fungsi utama dari *morphology kernel* adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang belum baik, penggunaan *morphology kernel* menjadi lebih optimal dan dapat meningkatkan akurasi pendeteksian.

Contoh citra data masukan pada uji coba lima ada pada Gambar 5.29. sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.30. Seperti yang dijelaskan sebelumnya setiap citra data masukan memiliki pasangan kunci jawabannya masing-masing. Kunci jawaban tersebut berisi citra yang sudah ditandai dan dipastikan benar pada bagian dari citra yang terkena serangan *copy-move.*

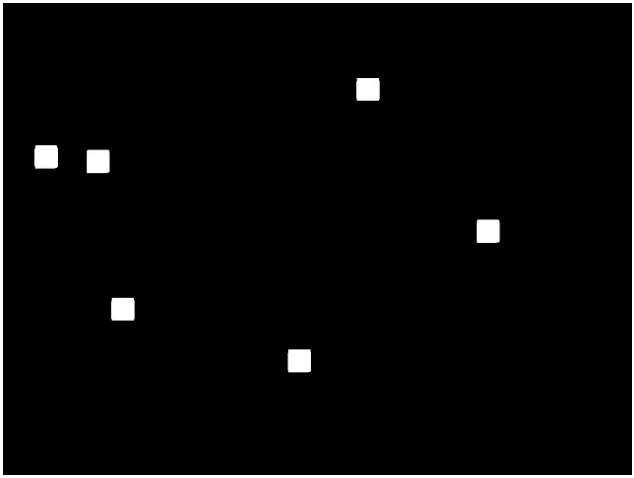
Gambar 5.29 Contoh citra masukan pada uji coba 5



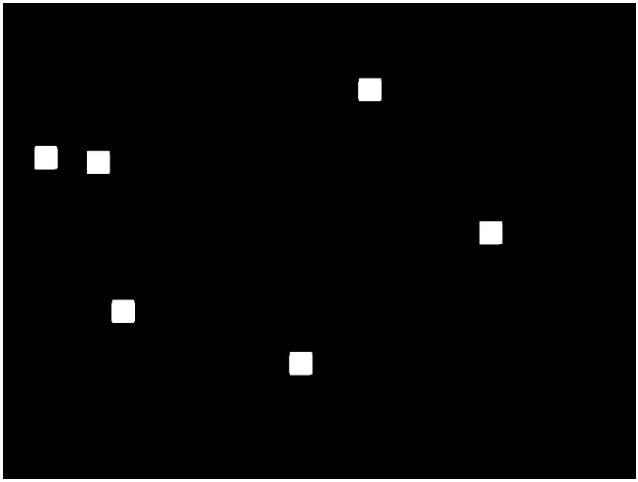
Gambar 5.30 Citra kunci jawaban pada uji coba 5

Hasil uji coba 5 pada citra data masukan pada Gambar 5.29 di atas menggunakan *kernel opening* ada pada Gambar 5.31, menggunakan *kernel closing* ada pada Gambar 5.32, menggunakan

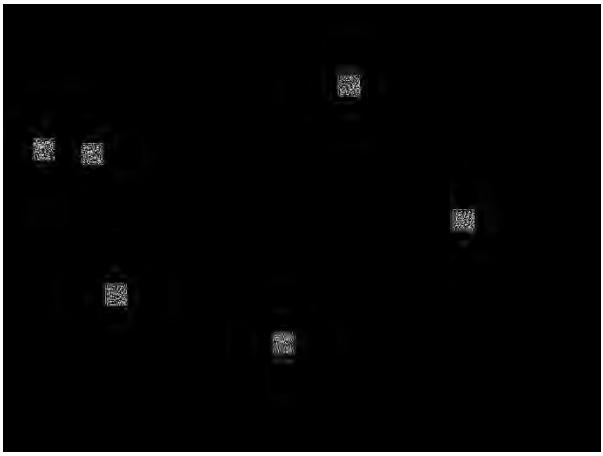
kernel edge ada pada Gambar 5.33, dan tanpa menggunakan *kernel* ada pada Gambar 5.34.



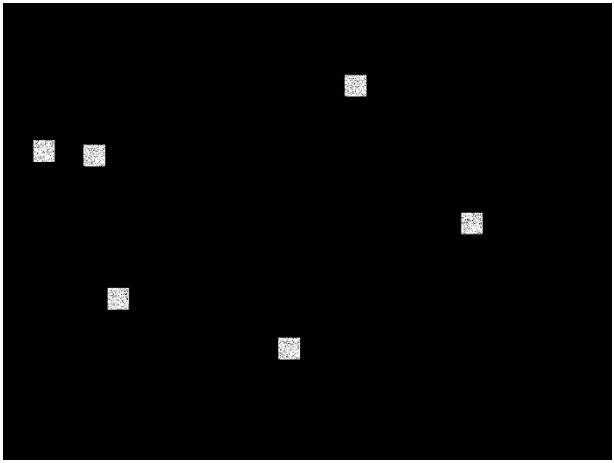
Gambar 5.31 Hasil uji coba 5 menggunakan *kernel opening*



Gambar 5.32 Hasil uji coba 5 menggunakan *kernel closing*



Gambar 5.33 Hasil uji coba 5 menggunakan *kernel edge*



Gambar 5.34 Hasil uji coba 5 tanpa menggunakan kernel

Hasil akhir dari uji coba ke-lima ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan metode

autocorrelation dan *Gaussian Filter.* Citra hasil keluaran pada uji coba ke 4 ada pada Gambar 5.35.

Gambar 5.35 Hasil akhir uji coba 5

5.4.6 Skenario Uji Coba 6

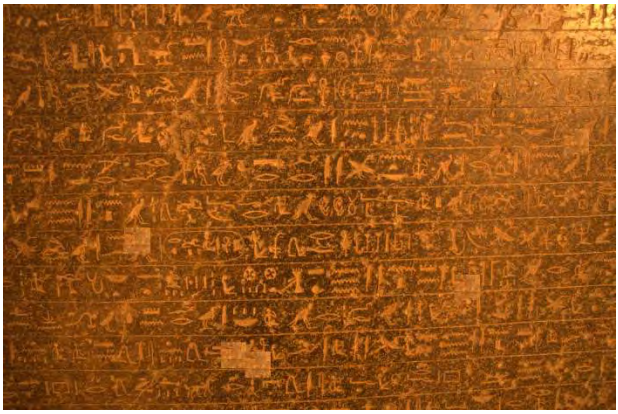
Skenario uji coba 6 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode modifikasi *exhaustive search*, tanpa menggunakan metode *Gaussian Filter* maupun *Laplacian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Jumlah dataset yang digunakan sebanyak 20 buah citra. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSE kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 5 dapat dilihat pada Tabel 5.7.

Tabel 5.7 Akurasi modifikasi metode *exhaustive* tanpa *Filter*

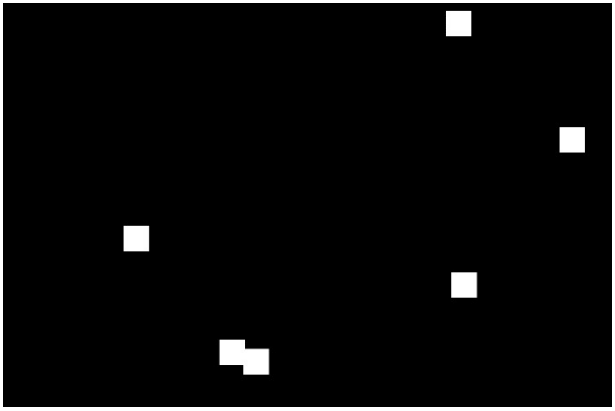
Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	16	4	**80%**
Kernel opening	10	10	50%
Kernel closing	11	9	55%

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, metode modifikasi *exhaustive search* tanpa *Filter*, memiliki akurasi tinggi apabila tidak menggunakan *kernel* yaitu mencapai **80%.** Sedangkan apabila menggunakan *kernel closing* hasil yang diperoleh juga hampir sama dengan *kernel opening* yaitu mencapai 55%. Akurasi mencapai nilai terendah apabila menggunakan *kernel opening* yaitu 50%. Nilai pada citra keluaran menggunakan *morphology kernel closing* dan *kernel opening* menghasilkan citra yang lebih menjauhi hasil keluaran yang diinginkan dibandingkan dengan citra yang tidak menggunakan *morphology*, disebabkan oleh citra yang diolah menggunakan *morphology kernel closing* dan *kernel opening* adalah citra tanpa *morphology* yang sudah menghasilkan citra keluaran yang sangat baik. Ketidakoptimalan hasil keluaran dari *morphology kernel* tersebut dikarenakan kernel *close* tidak menandai bagian/area dari hasil keluaran metode *autocorrelation*, karena sudah dianggap sesuai dengan hasil yang diharapkan. Fungsi utama dari *morphology kernel* adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang sudah baik, penggunaan *morphology kernel* menjadi tidak optimal.

Contoh citra data masukan pada uji coba enam ada pada Gambar 5.36 Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.37.

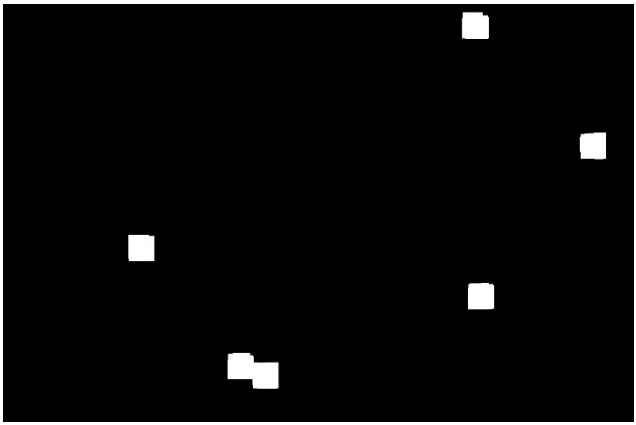


Gambar 5.36 Contoh citra masukan pada uji coba 6



Gambar 5.37 Citra kunci jawaban pada uji coba 6

Hasil uji coba 6 pada citra data masukan pada Gambar 5.36 di atas menggunakan *kernel opening* ada pada Gambar 5.38, menggunakan *kernel closing* ada pada Gambar 5.39, menggunakan *kernel edge* ada pada Gambar 5.40, dan tanpa menggunakan *kernel* ada pada Gambar 5.41.

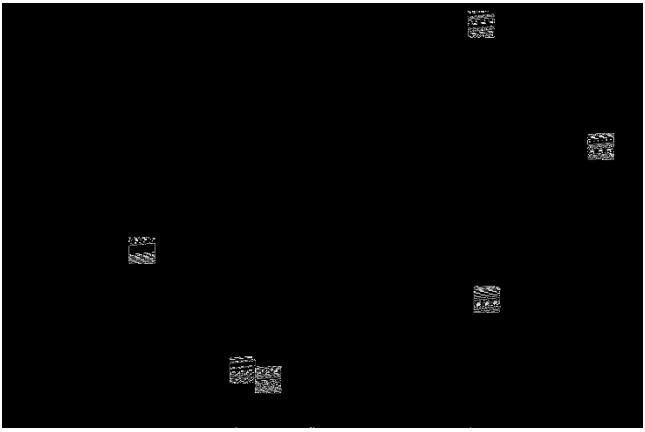


Gambar 5.38 Hasil uji coba 6 menggunakan *kernel opening*

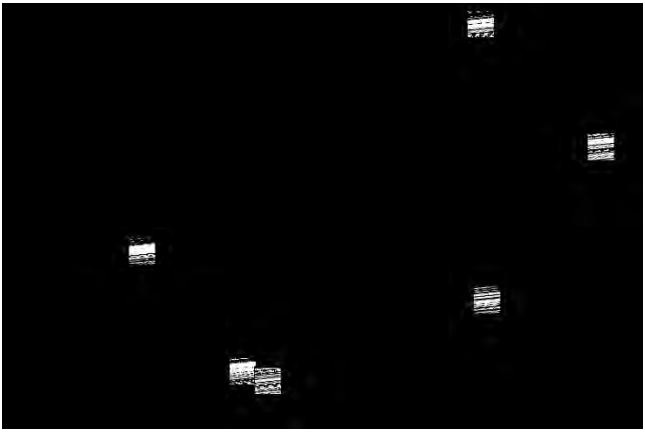
Gambar 5.39 Hasil uji coba 6 menggunakan *kernel closing*

Hasil pada Gambar 5.40 dapat memperbaiki hasil pada citra keluaran tanpa *morphology kernel* pada Gambar 5.41 yang disebabkan oleh *morphology kernel closing* dapat dengan baik mengolah citra keluaran dari *morphology edge detection*/*Canny.*

Pada Gambar 5.41 dapat dilihat bahwa letak *edge* yang berdekatan pada setiap objek diperkirakan terkena serangan *copy-move*.



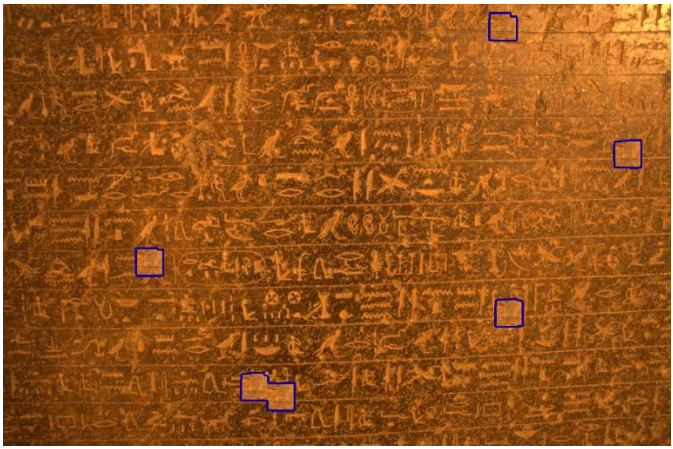
Gambar 5.40 Hasil uji coba 6 menggunakan *kernel edge*



Gambar 5.41 Hasil uji coba 6 tanpa menggunakan *morphology kernel*

Hasil akhir dari uji coba ke-enam ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa

kernel untuk mendeteksi *copy-move* menggunakan modifikasi metode *exhaustive* dan tanpa metode *Filter.* Citra hasil keluaran pada uji coba ke 6 ada pada Gambar 5.42.



Gambar 5.42 Hasil akhir uji coba 6

5.4.7 Skenario Uji Coba 7

Skenario uji coba 7 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode modifikasi metode *exhaustive search*, dan mengimplementasikan *preprocessing* menggunakan metode *Gaussian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%.

Jumlah dataset yang digunakan sebanyak 20 buah citra. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSEnya kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 7 dapat dilihat pada Tabel 5.8.

Tabel 5.8 Akurasi modifikasi metode *exhaustive* dan *Gaussian Filter*

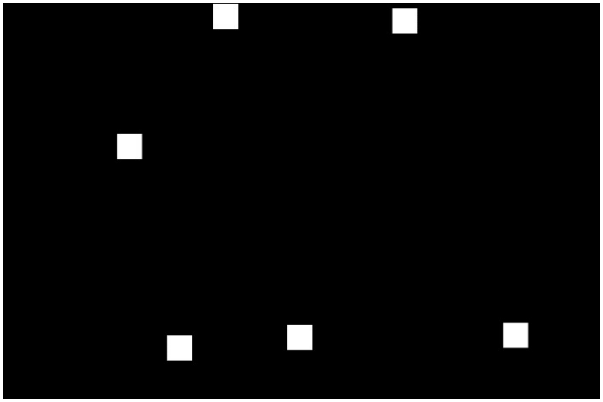
Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	18	2	**90%**
Kernel opening	11	9	55%
Kernel closing	12	8	60%

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, modifikasi metode *exhaustive search* menggunakan *Gaussian Filter*, memiliki akurasi tinggi apabila tidak menggunakan *kernel* yaitu mencapai **90%.** Sedangkan apabila menggunakan *kernel closing* hasil yang diperoleh juga hampir sama dengan *kernel opening* yaitu mencapai 60%. Akurasi mencapai nilai terendah apabila menggunakan *kernel opening* yaitu 55%. Nilai pada citra keluaran pada uji coba 7 tanpa *morphology kernel* dapat dengan baik mendapatkan akurasi yang tinggi dibandingkan dengan metode *exhaustive* pada uji coba pertama, karena adanya perubahan pada metode *exhaustive search.*

Hasil keluaran dari modifikasi metode *exhaustive* dengan *Gaussian Filter* yang menggunakan *kernel opening* dan *kernel closing* sangat jauh dari hasil yang diharapkan. Hal ini dikarenakan apabila penggunaan *morphology kernel* diimplementasikan pada hasil keluaran yang sudah baik dari modifikasi metode *exhaustive*, maka hasil keluaran dari *morphology kernel* menjadi tidak optimal. Ketidakoptimalan hasil keluaran dari *morphology kernel* tersebut dikarenakan kernel *close* tidak menandai bagian/area dari hasil keluaran modifikasi metode *exhaustive*, karena sudah dianggap sesuai dengan hasil yang diharapkan. Fungsi utama dari morphology kernel adalah menandai bagian dari hasil keluaran modifikasi metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran modifikasi metode *exhaustive* yang sudah baik, penggunaan *morphology kernel* menjadi tidak optimal.

Contoh citra data masukan pada uji coba tujuh ada pada Gambar 5.43. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.44.

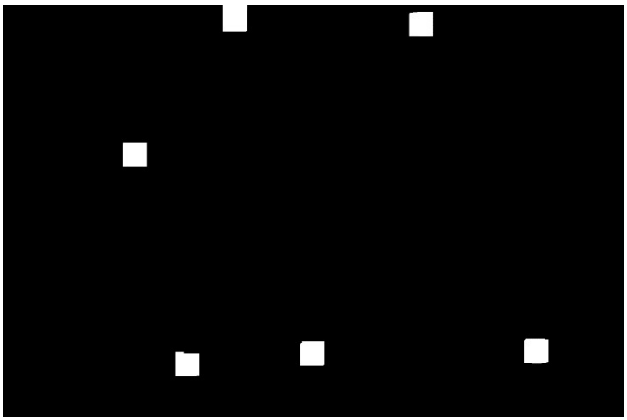
Gambar 5.43 Contoh citra masukan pada uji coba 7



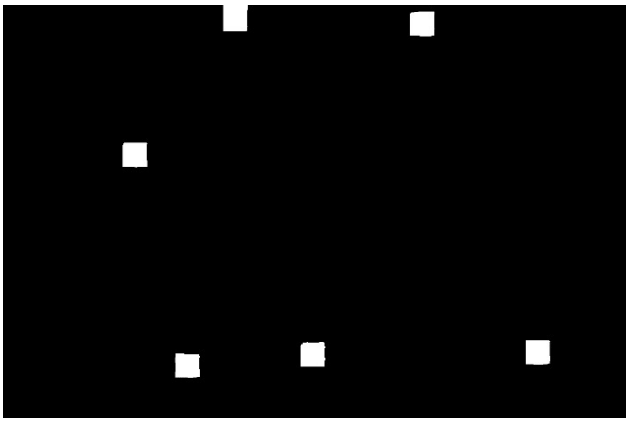
Gambar 5.44 Citra kunci jawaban pada uji coba 7

Hasil uji coba 7 pada citra data masukan pada Gambar 5.43 di atas menggunakan *kernel opening* ada pada Gambar 5.45, menggunakan *kernel closing* ada pada gambar 5.46, menggunakan

kernel edge ada pada Gambar 5.47, dan tanpa menggunakan *kernel* ada pada Gambar 5.48.



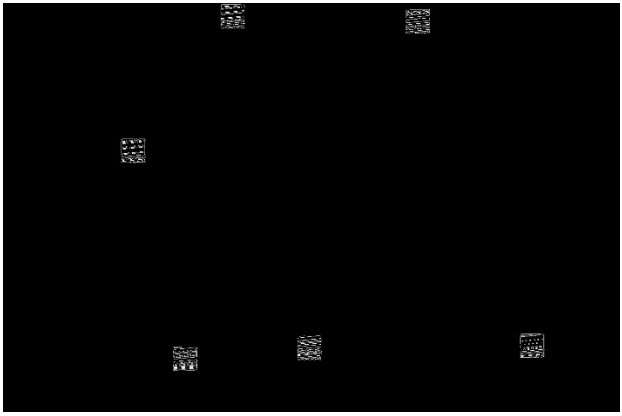
Gambar 5.45 Hasil uji coba 7 menggunakan *kernel opening*



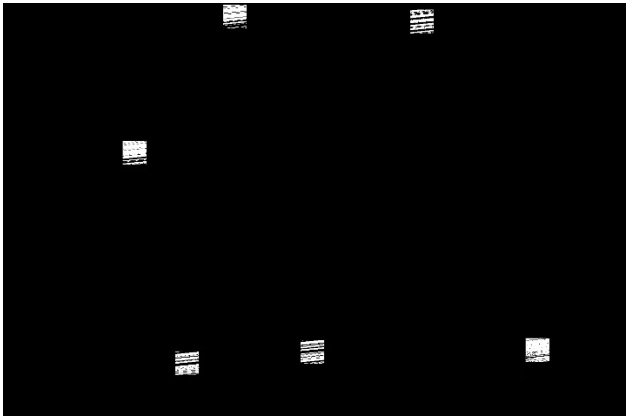
Gambar 5.46 Hasil uji coba 7 menggunakan *kernel closing*

Hasil pada Gambar 5.45 adalah hasil gambar yang dihasilkan dari proses citra pada Gambar 5.46. Pada kedua gambar

hanya terdapat sedikit perbedaan dikarenakan tidak banyaknya *noise* yang dihasilkan dari modifikasi metode *exhaustive search*.



Gambar 5.47 Hasil uji coba 7 menggunakan *kernel edge*



Gambar 5.48 Hasil uji coba 7 tanpa menggunakan *morphology kernel*

Hasil akhir dari uji coba ke-tujuh ini diambil dari hasil terbaik antara penggunaan *kernel open*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan modifikasi

metode *exhaustive* dan *Gaussian Filter.* Citra hasil keluaran pada uji coba ke 7 ada pada Gambar 5.49.

Gambar 5.49 Hasil akhir uji coba 7

5.4.8 Skenario Uji Coba 8

Skenario uji coba 8 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode modifikasi metode *exhaustive search*, dan mengimplementasikan *preprocessing* menggunakan metode *Laplacian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Jumlah dataset yang digunakan sebanyak 20 buah citra. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSEnya kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 8 dapat dilihat pada Tabel 5.9.

Tabel 5.9 Akurasi modifikasi metode *exhaustive* dan *Laplacian Filter*

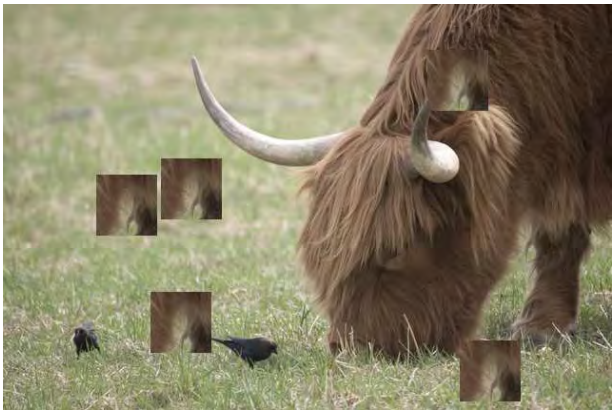
Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	11	9	55%
Kernel opening	18	2	**90%**
Kernel closing	18	2	**90%**

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, modifikasi metode *exhaustive search* menggunakan *Laplacian Filter*, memiliki akurasi tinggi apabila menggunakan *kernel opening* atau *kernel closing* yaitu mencapai **90%.** Sedangkan akurasi mencapai nilai terendah apabila tanpa menggunakan *kernel* yaitu 55%.

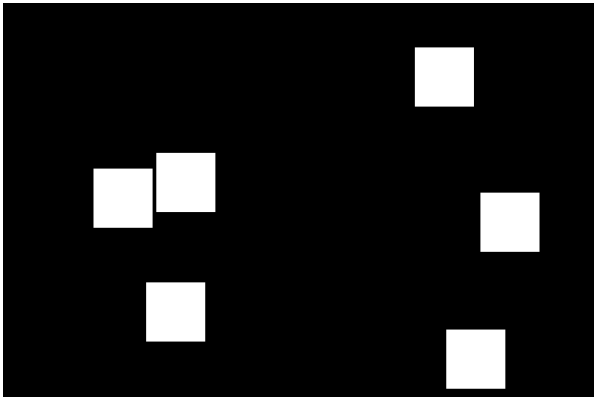
Dari hasil yang dilakukan uji akurasi pada uji coba 8 dapat diketahui bahwa citra tanpa *morphology kernel* memperoleh hasil yang kurang baik namun objek yang dideteksi sudah dapat dengan mudah diperbaiki oleh *morphology kernel,* baik *closing* maupun *opening*. Pola objek yang diprediksi oleh pemrosesan citra tanpa *kernel* menghasilkan pola objek yang berupa kumpulan piksel yang berdekatan sehingga mudah untuk diolah menggunakan *morphology kernel* kemudian menghasilkan citra keluaran yang mirip dengan citra kunci jawaban.

Fungsi utama dari *morphology kernel* adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang belum baik, penggunaan *morphology kernel* menjadi lebih optimal dan dapat meningkatkan akurasi pendeteksian.

Contoh citra data masukan pada uji coba delapan ada pada Gambar 5.50. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.51.

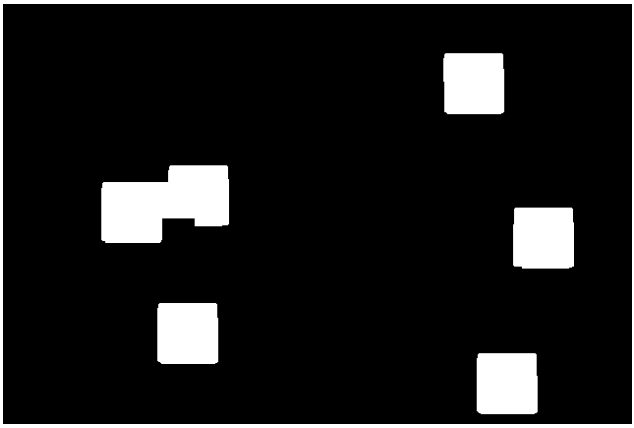


Gambar 5.50 Contoh citra masukan pada uji coba 8

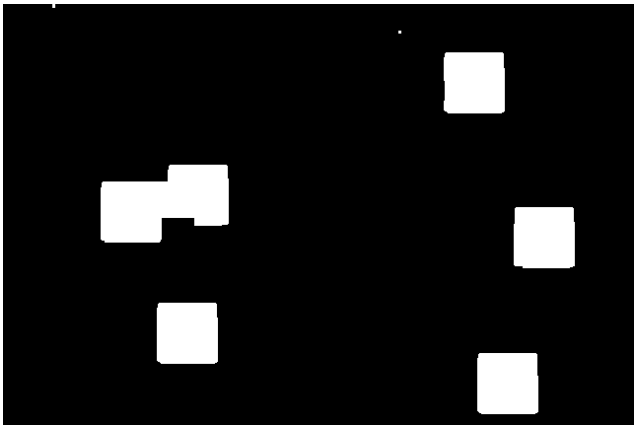


Gambar 5.51 Citra kunci jawaban pada uji coba 8

Hasil uji coba 8 pada citra data masukan pada Gambar 5.50 di atas menggunakan *kernel opening* ada pada Gambar 5.52, menggunakan *kernel closing* ada pada gambar 5.53, menggunakan *kernel edge* ada pada Gambar 5.54, dan tanpa menggunakan *kernel* ada pada Gambar 5.55.

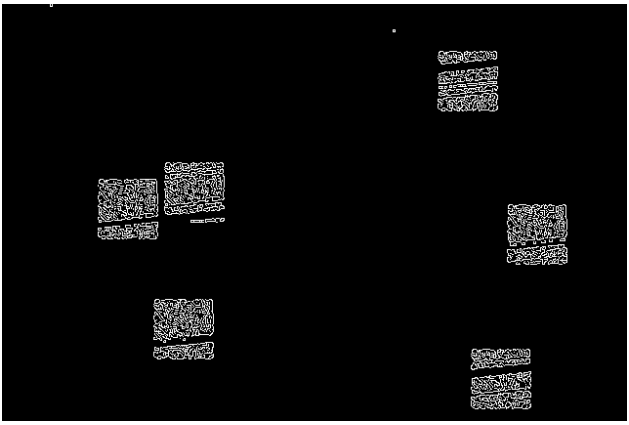


Gambar 5.52 Hasil uji coba 8 menggunakan *kernel opening*

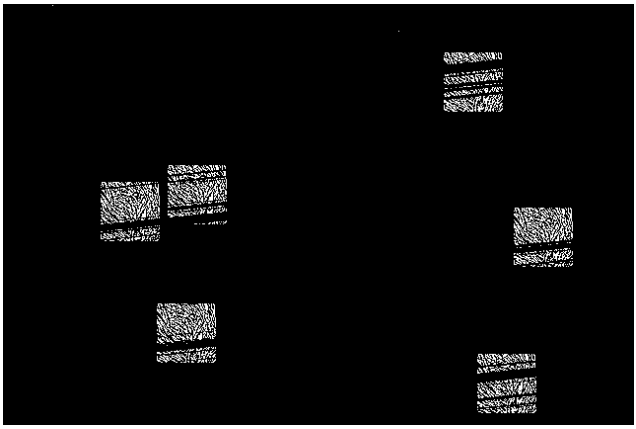


Gambar 5.53 Hasil uji coba 8 menggunakan *kernel closing*

Pada Gambar 5.52 dapat dilihat bahwa adanya peningkatan kualitas keluaran citra yang lebih mendekati dengan citra kunci jawaban dibandingkan pada Gambar 5.53 karena adanya konversi beberapa jumlah *noise* menjadi *background*/piksel hitam pada proses morphology *kernel opening*.

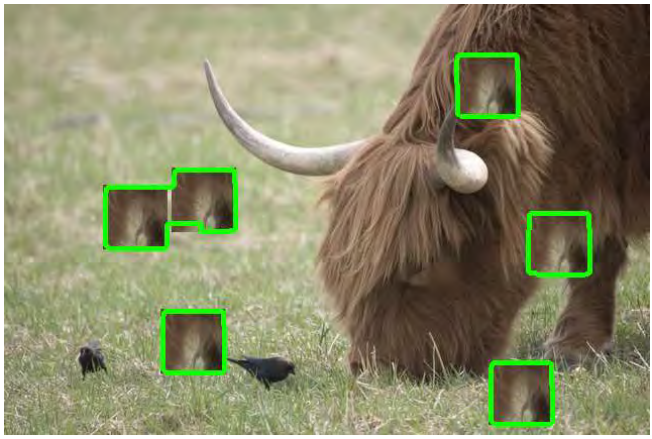


Gambar 5.54 Hasil uji coba 8 menggunakan *kernel edge*



Gambar 5.55 Hasil uji coba 8 tanpa menggunakan *kernel morphology*

Hasil akhir dari uji coba ke-delapan ini diambil dari hasil terbaik antara penggunaan *kernel opening*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan modifikasi metode *exhaustive* dan *Laplacian Filter.* Citra hasil keluaran pada uji coba ke 8 ada pada Gambar 5.56.



Gambar 5.56 Hasil akhir uji coba 8

5.4.9 Skenario Uji Coba 9

Skenario uji coba 9 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan modifikasi metode *autocorrelation*, dan mengimplementasikan *preprocessing* menggunakan metode *Gaussian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Jumlah dataset yang digunakan sebanyak 20 buah citra. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSEnya kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 9 dapat dilihat pada Tabel 5.10.

Tabel 5.10 Akurasi modifikasi metode *autocorrelation* modifikasi menggunakan *Gaussian Filter*

Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	13	7	65%

Kernel opening	17	3	**85%**
Kernel closing	17	3	**85%**

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, *modifikasi autocorrelation* menggunakan *Gaussian Filter*, memiliki akurasi tinggi apabila menggunakan *kernel opening* atau *kernel closing* yaitu mencapai **90%.** Sedangkan akurasi mencapai nilai terendah apabila tanpa menggunakan *kernel* yaitu 65%.

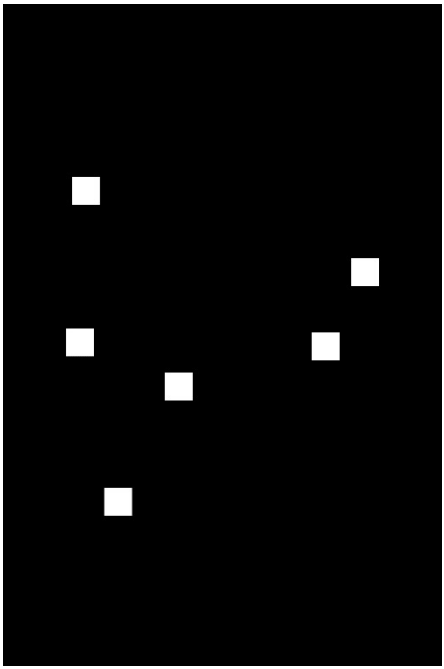
Dari hasil yang dilakukan uji akurasi pada uji coba 9 dapat diketahui bahwa citra tanpa *morphology kernel* memperoleh hasil yang kurang baik namun objek yang dideteksi sudah dapat dengan mudah diperbaiki oleh *morphology kernel,* baik *closing* maupun *opening*. Pola objek yang diprediksi oleh pemrosesan citra tanpa *kernel* menghasilkan pola objek yang berupa kumpulan piksel yang berdekatan sehingga mudah untuk diolah menggunakan *morphology kernel* kemudian menghasilkan citra keluaran yang mirip dengan citra kunci jawaban.

Fungsi utama dari *morphology kernel* adalah menandai bagian dari hasil keluaran metode *exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *exhaustive* yang belum baik, penggunaan *morphology kernel* menjadi lebih optimal dan dapat meningkatkan akurasi pendeteksian.

Contoh citra data masukan pada uji coba sembilan ada pada Gambar 5.57. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.58. Seperti yang dijelaskan sebelumnya setiap citra data masukan memiliki pasangan kunci jawabannya masing-masing. Kunci jawaban tersebut berisi citra yang sudah ditandai dan dipastikan benar pada bagian dari citra yang terkena serangan *copy-move.*

Gambar 5.57 Contoh citra masukan pada uji coba 9

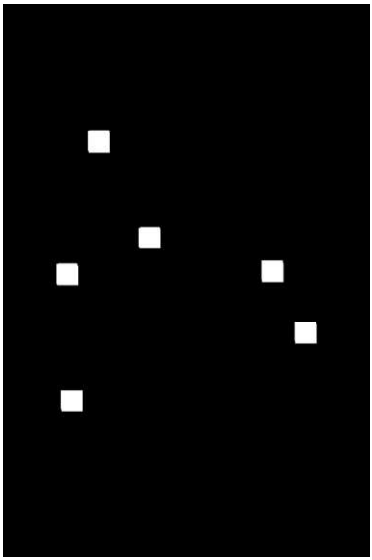
Hasil uji coba 9 pada citra data masukan pada Gambar 5.57 di atas menggunakan *kernel opening* ada pada Gambar 5.59, menggunakan *kernel closing* ada pada gambar 5.60, menggunakan *kernel edge* ada pada Gambar 5.61, dan tanpa menggunakan *kernel* ada pada Gambar 5.62.



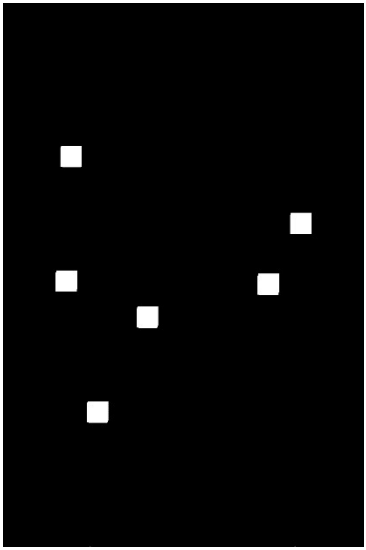
Gambar 5.58 Citra kunci jawaban pada uji coba 9

Pada Gambar 5.59 dapat dilihat bahwa hampir sama dengan citra pada Gambar 5.60. Namun ada sedikit peningkatan kualitas keluaran citra yang lebih mendekati dengan citra kunci jawaban dibandingkan pada Gambar 5.60 karena adanya konversi beberapa jumlah *noise* menjadi *background*/piksel hitam pada proses morphology *kernel opening*.

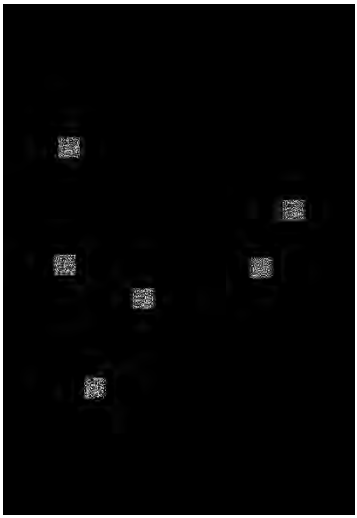
Selain itu *kernel opening* pada Gambar 5.59 dibentuk dari hasil keluaran *kernel closing* pada Gambar 5.60, sehingga hasil *kernel opening* adalah menyempurnakan hasil keluaran dari *kernel closing*. Oleh karena itu hasil pada Gambar 5.59 sedikit lebih baik daripada citra pada Gambar 5.60.



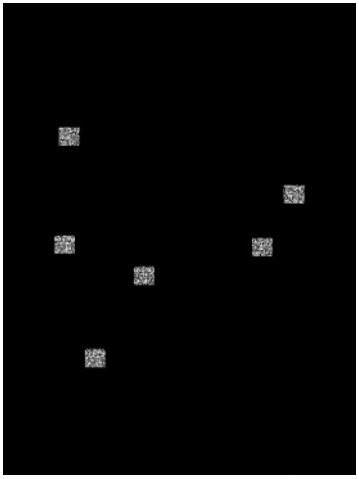
Gambar 5.59 Hasil uji coba 9 menggunakan *kernel opening*



Gambar 5.60 Hasil uji coba 9 menggunakan *kernel closing*

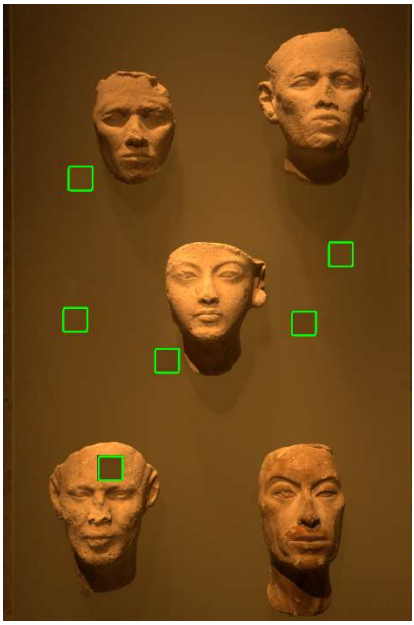


Gambar 5.61 Hasil uji coba 9 menggunakan *kernel edge*



Gambar 5.62 Hasil uji coba 9 tanpa menggunakan *kernel morphology*

Hasil akhir dari uji coba ke-sembilan ini diambil dari hasil terbaik antara penggunaan *kernel opening*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan modifikasi metode *autocorrelation* dan *Gaussian Filter.* Citra hasil keluaran pada uji coba ke 9 ada pada Gambar 5.63.



Gambar 5.63 Hasil akhir uji coba 9

5.4.10 Skenario Uji Coba 10

Skenario uji coba 10 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan modifikasi metode *autocorrelation*, dan mengimplementasikan *preprocessing* menggunakan metode *Laplacian Filter.* Skenario dilakukan dengan menerapkan penggunaan *kernel opening*, *kernel closing*, dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan

100%. Jumlah dataset yang digunakan sebanyak 20 buah citra. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSEnya kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 10 dapat dilihat pada Tabel 5.11.

Tabel 5.11 Akurasi modifikasi metode *autocorrelation* menggunakan *Laplacian Filter*

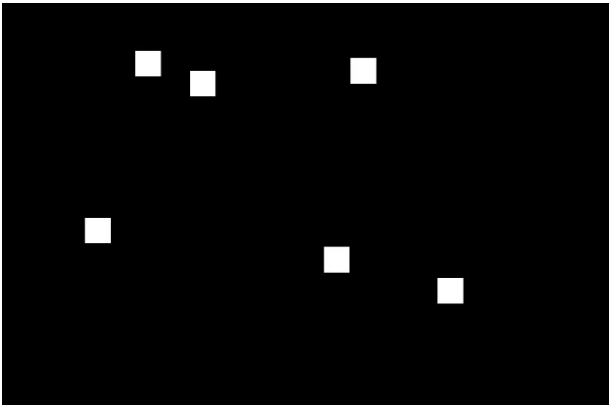
Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	3	17	15%
Kernel opening	15	5	**75%**
Kernel closing	15	5	**75%**

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, *modifikasi autocorrelation* menggunakan *Laplacian Filter*, memiliki akurasi tinggi apabila menggunakan *kernel opening* atau *kernel closing* yaitu mencapai **75%.** Sedangkan akurasi mencapai nilai terendah apabila tanpa menggunakan *kernel* yaitu 15%.

Dari hasil yang dilakukan uji akurasi pada uji coba 10 dapat diketahui bahwa citra tanpa *morphology kernel* memperoleh hasil yang kurang baik namun objek yang dideteksi sudah dapat dengan mudah diperbaiki oleh *morphology kernel,* baik *closing* maupun *opening*. Pola objek yang diprediksi oleh pemrosesan citra tanpa *kernel* menghasilkan pola objek yang berupa kumpulan piksel yang berdekatan sehingga mudah untuk diolah menggunakan *morphology kernel* kemudian menghasilkan citra keluaran yang mirip dengan citra kunci jawaban.

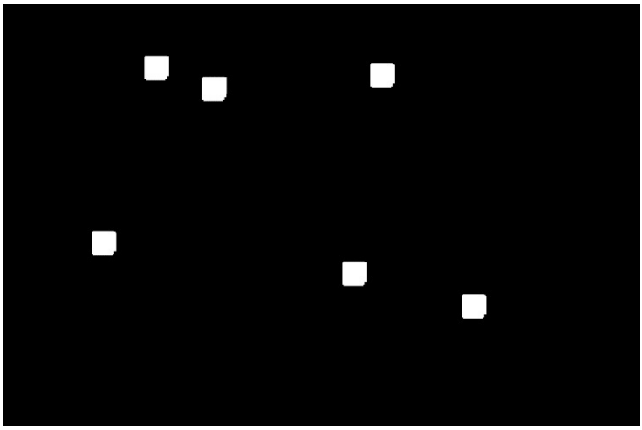
Contoh citra data masukan pada uji coba sepuluh ada pada Gambar 5.64. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.65.

Gambar 5.64 Contoh citra masukan pada uji coba 10

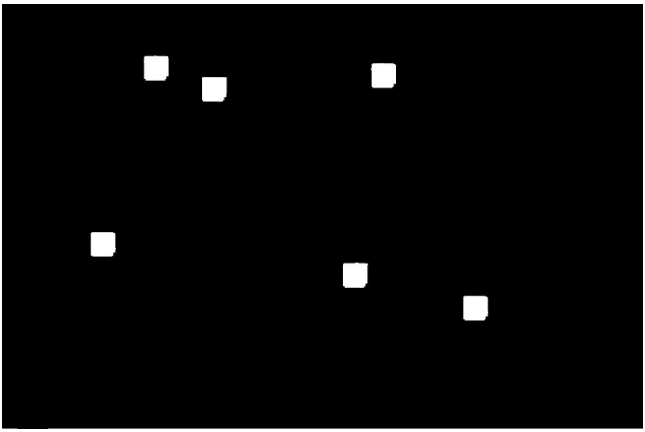


Gambar 5.65 Citra kunci jawaban pada uji coba 10

Hasil uji coba 10 pada citra data masukan pada Gambar 5.64 di atas menggunakan *kernel opening* ada pada Gambar 5.66, menggunakan *kernel closing* ada pada gambar 5.67, menggunakan *kernel edge* ada pada Gambar 5.68, dan tanpa menggunakan *kernel* ada pada Gambar 5.69.

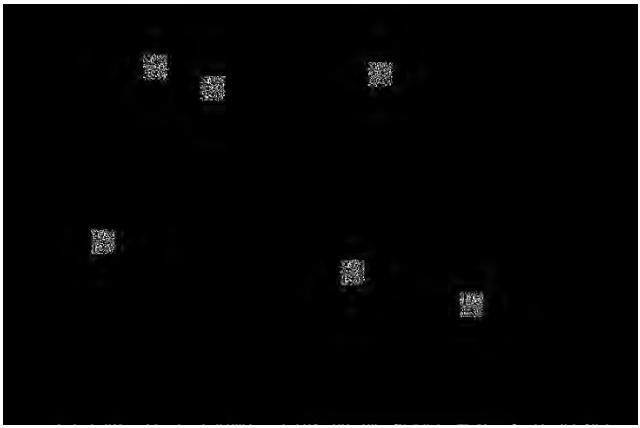


Gambar 5.66 Hasil uji coba 10 menggunakan *kernel opening*



Gambar 5.67 Hasil uji coba 10 menggunakan *kernel closing*

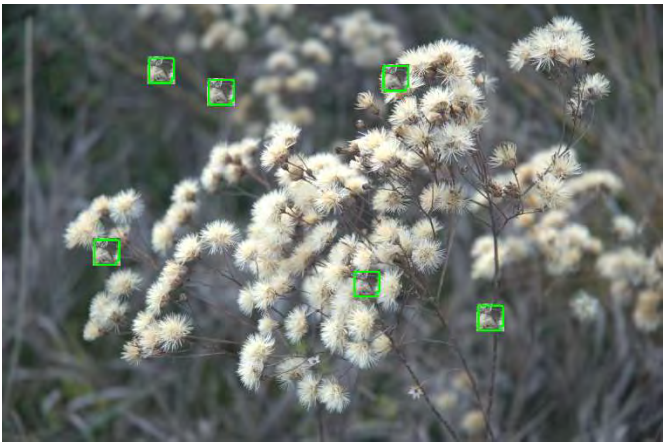
Hasil keluaran citra pada Gambar 5.67 dapat dengan baik mengolah citra pada *morphology kernel edge* yang sudah dengan baik mengolah dari keluaran citra tanpa *kernel* yang dapat memprediksi lokasi-lokasi yang diperkirakan terkena serangan *copy-move.*



Gambar 5.68 Hasil uji coba 10 menggunakan *kernel edge*

Gambar 5.69 Hasil uji coba 10 tanpa menggunakan *kernel morphology*

Hasil akhir dari uji coba ke-sepuluh ini diambil dari hasil terbaik antara penggunaan *kernel opening*, *kernel closing*, atau tanpa kernel untuk mendeteksi *copy-move* menggunakan modifikasi metode *autocorrelation* dan *Laplacian Filter.* Citra hasil keluaran pada uji coba ke 10 ada pada Gambar 5.70.



Gambar 5.70 Hasil akhir uji coba 10

5.4.11 Skenario Uji Coba 11

Skenario uji coba 11 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *naive exhaustive*. metode *naive exhaustive* merupakan metode asli dari exhaustive yang tidak dimodifikasi sama sekali. Skenario dilakukan dengan menerapkan penggunaan *kernel opening* dan tanpa penggunaan *kernel*. Nilai akurasi diperoleh dari hasil pembagian antara nilai *true* dengan jumlah dataset dan dikalikan 100%. Jumlah dataset yang digunakan sebanyak 20 buah citra. Nilai *true* sendiri diperoleh dari jumlah citra yang nilai MSEnya kurang dari nilai MSE *black* citra yang digunakan dalam uji coba pendeteksian *copy-move* citra. Hasil akurasi pada uji coba 11 dapat dilihat pada Tabel 5.12.

Tabel 5.12 Akurasi metode *naive exhaustive*

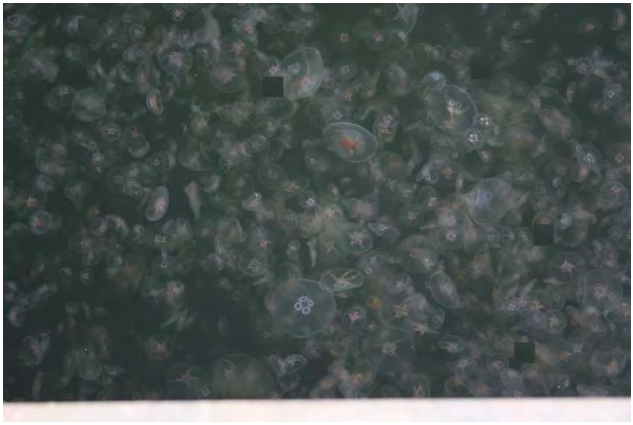
Metode	Hasil		Akurasi
	True	*False*	
Tanpa *kernel*	14	6	**70%**
Kernel opening	13	7	65%

Berdasarkan hasil yang ditunjukkan pada penghitungan akurasi diatas, *naive exhaustive* memiliki akurasi tinggi apabila tanpa menggunakan suatu kernel yaitu mencapai **70%.** Sedangkan akurasi mencapai nilai terendah apabila menggunakan *kernel opening* yaitu 65%.

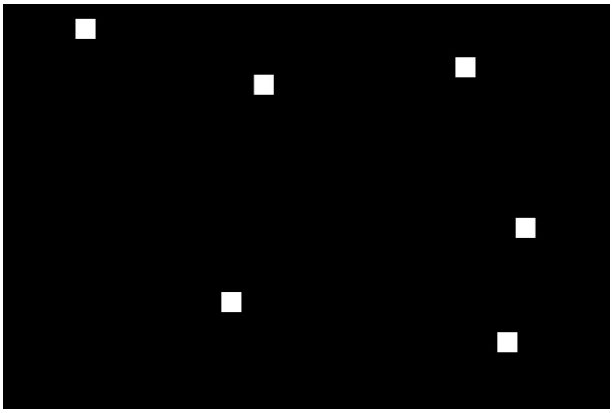
Hasil keluaran metode naive *exhaustive* yang menggunakan *kernel opening* menjauhi dari hasil yang diharapkan. Hal ini dikarenakan apabila penggunaan *morphology kernel* diimplementasikan pada hasil keluaran yang sudah baik dari metode *naive exhaustive*, maka hasil keluaran dari *morphology kernel* menjadi tidak optimal.

Fungsi utama dari morphology kernel adalah menandai bagian dari hasil keluaran metode *naive exhaustive* yang belum sempurna/sesuai dengan yang diharapkan. Sehingga apabila *morphology kernel* diterapkan pada citra hasil keluaran metode *naive exhaustive* yang sudah baik, penggunaan *morphology kernel* menjadi tidak optimal.

Contoh citra data masukan pada uji coba sebelas ada pada Gambar 5.71. Sedangkan kunci jawaban dari citra tersebut ada pada Gambar 5.72.

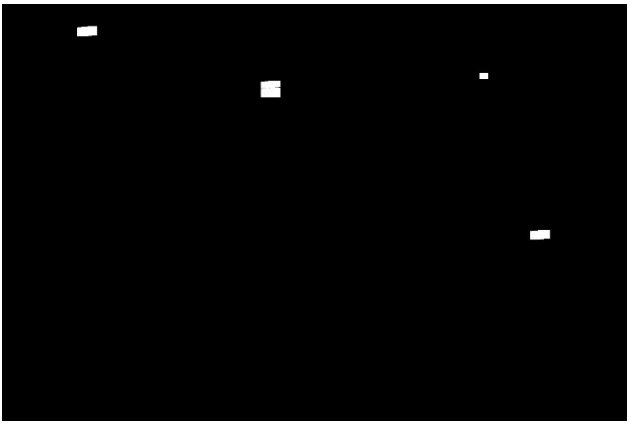


Gambar 5.71 Contoh citra masukan pada uji coba 11

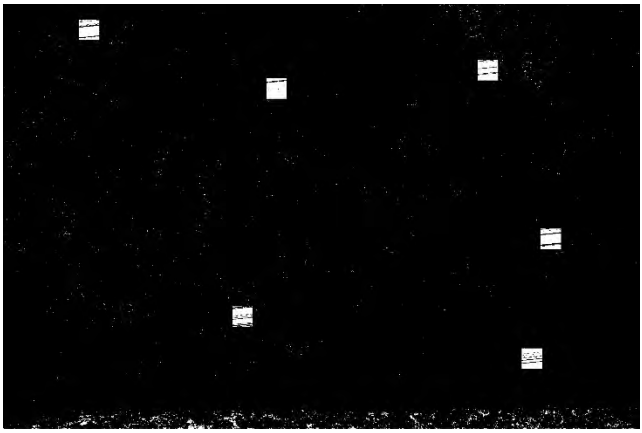


Gambar 5.72 Citra kunci jawaban pada uji coba 11

Hasil uji coba 11 pada citra data masukan pada Gambar 5.71 di atas menggunakan *kernel opening* ada pada Gambar 5.73, dan tanpa menggunakan *kernel* ada pada Gambar 5.74.

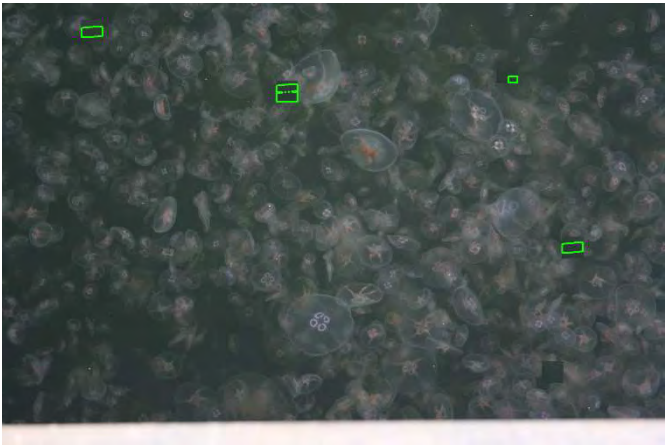


Gambar 5.73 Hasil uji coba 11 menggunakan *kernel opening*



Gambar 5.74 Hasil uji coba 11 tanpa menggunakan *kernel*

Hasil akhir dari uji coba ke-sebelas ini diambil dari hasil terbaik antara penggunaan *kernel opening* atau tanpa kernel untuk mendeteksi *copy-move* menggunakan metode *naive exhaustive.* Citra hasil keluaran pada uji coba ke 11 ada pada Gambar 5.75.



Gambar 5.75 Hasil akhir uji coba 11

5.5 Evaluasi Umum Skenario Uji Coba

Berdasarkan skenario uji coba ke satu sampai lima yang telah dilakukan, dapat diketahui bahwa tingkat ketepatan sistem dalam mendeteksi *copy-move* pada citra sudah sangat baik dengan tingkat akurasi mencapai **88,57%**, baik menggunakan metode *exhaustive* maupun *autocorrelation*. Dari hasil uji coba juga diketahui bahwa *preprocessing* dengan menggunakan suatu metode *Filter* dapat meningkatkan akurasi dan ketepatan pendeteksian *copy-move* pada citra. Dari ke lima uji coba tersebut dapat disimpulkan bahwa metode *Gaussian Filter* lebih baik digunakan dalam tahap preprocessing dibandingkan dengan *Laplacian Filter.*

Sedangkan untuk penggunaan *kernel*, dari hasil uji coba di atas, metode *exhaustive search* memiliki tingkat akurasi tinggi apabila tidak menggunakan *kernel*, karena apabila metode *exhaustive search* diterapkan dengan penggunaan *kernel* hasil akurasi yang diperoleh mendekati **0%**. Sedangkan metode *autocorrelation* memiliki tingkat akurasi yang tinggi apabila menggunakan suatu *kernel*. Dari uji coba di atas *kernel opening* terbukti lebih baik dari pada *kernel closing* pada penggunaan metode *autocorrelation*.

Sedangkan berdasarkan skenario uji coba ke enam sampai sepuluh yang telah dilakukan, dapat diketahui bahwa tingkat ketepatan sistem dalam mendeteksi *copy-move* pada citra sudah sangat baik dengan tingkat akurasi mencapai **90%**, baik menggunakan modifikasi metode *exhaustive* maupun modifikasi metode *autocorrelation*. Dari hasil uji coba juga diketahui bahwa Preprocessing dengan menggunakan suatu metode *Filter* dapat meningkatkan akurasi dan ketepatan pendeteksian *copy-move* pada citra. Dari ke lima uji coba tersebut dapat disimpulkan bahwa metode *Gaussian Filter* lebih baik digunakan dalam tahap *preprocessing* dibandingkan dengan *Laplacian Filter.*

Sedangkan untuk penggunaan *kernel*, dari hasil uji coba ke eneam sampai sepuluh di atas, modifikasi metode *exhaustive* memiliki tingkat akurasi tinggi apabila tidak menggunakan *kernel* atau menggunakan *kernel opening*. Sedangkan metode modifikasi *autocorrelation* memiliki tingkat akurasi yang tinggi apabila menggunakan suatu *kernel*. Dari uji coba di atas *kernel opening* maupun *kernel closing* dapat meningkatkan akurasi pada penggunaan modifikasi metode *autocorrelation*.

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan perangkat lunak nantinya.

6.1 Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil uji coba pendeteksian *copy-move* citra menggunakan metode *exhaustive* dan *autocorrelation* adalah sebagai berikut:

1. Implementasi metode *Exhasutive* dan *autocorrelation* dapat digunakan sebagai salah satu metode untuk mendeteksi dan menganalisis serangan *copy-move* pada suatu citra.
2. Implementasi *Gaussian dan laplacian Filter* dapat dijadikan metode yang cocok untuk menghilangkan atau meminimalisasi *noise* yang ada pada citra.
3. Implementasi *kernel morphology* dalam pendeteksian serangan *copy-move* pada citra dapat meningkatkan ketepatan pendeteksian pada metode *autocorrelation*. Namun juga dapat menurunkan akurasi dan ketepatan pendeteksian.
4. Pada kasus Tugas Akhir ini, akurasi terbaik didapat dengan pendeteksian menggunakan modifikasi metode *exhaustive* dengan *Gaussian Filter* dan tanpa menggunakan *kernel*.
5. Akurasi tertinggi yang didapat dari kesepeuluh skenario uji coba adalah 90%.
6. Modifikasi pada metode *exhaustive* maupun *autocorrelation* terbukti dapat meningkatkan akurasi sistem dalam mendeteksi serangan *copy-move* pada citra, yaitu mencapai 90%.

7. Metode *exhaustive search* maupun *autocorrelation* baik untuk mendeteksi serangan copy-move di dalam citra pada objek yang besar.

6.2 Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah:

1. Menambah jumlah data karena dari total 35 data citra dan 20 data citra dianggap masih terlalu sedikit.
2. Dapat dicoba menggunakan suatu metode segmentasi dan klasifikasi untuk meningkatkan ketepatan pendeteksian serangan *copy-move* pada citra.

DAFTAR PUSTAKA

[1] J. Fridrich, D. Soukal dan J. Lukas, "Detection of Copy-Move Foergery in Digital Images," *Proceedings of Digital Forensic Research Workshop,* 2002.

[2] R. Judith, T. Wiem dan D. Jean-Luc, "Digital image forensics: a booklet for beginners," *Multimedia Tools and Applications,* vol. 51, pp. 133-162, 2011.

[3] "Anaconda | Continuum Analytics: Documentation," Anaconda, [Online]. Available: https://docs.continuum.io/anaconda/index. [Diakses 16 June 2016].

[4] "OpenCV," OPEN SOURCE COMPUTER VISION, [Online]. Available: http://opencv.org/. [Diakses 16 June 2016].

[5] S. Community, "NumPy v1.10 Manual," 18 October 2015. [Online]. Available: http://docs.scipy.org/doc/numpy-1.10.1/index.html. [Diakses May 2016].

[6] "SciPy," Scientific Computing Tools for Python, [Online]. Available: https://scipy.org/. [Diakses 16 June 2016].

[7] "SQLite," SQLite, [Online]. Available: https://www.sqlite.org/. [Diakses 16 June 2016].

[8] "SQLite3 Library DB API," SQLite3, [Online]. Available: https://docs.python.org/2/library/sqlite3.html. [Diakses 16 June 2016].

[9] M. Alam dan S. Ray, "Design of an Intelligent SHA-1 Based," *International Journal of Network Security,* vol. 15, pp. 465-470, 2013.

[10] "Scikit-Learn," Scikit-Learn, [Online]. Available: http://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error. [Diakses 16 June 2016].

[11] M. Reith, M. Reith, C. Carr dan G. Gunsh, "An Examination of Digital Forensic Models," *International Journal of Digital Evidence,* vol. 1, no. 3, 2002.

[12] H. Farid, Digital Image Forensics, Dartmouth.

BIODATA PENULIS

Yusuf Nugroho merupakan anak dari pasangan Bapak Sudiro dan Ibu Cholifah. Lahir di Surabaya pada tanggal 20 Februari 1995. Penulis menempuh pendidikan formal dimulai dari TK Panca Putra (1999-2000), SDN Kalisari 05 Jakarta (2000-2006), SMPN 103 Jakarta (2006-2009), SMAN 98 Jakarta (2009-2012) dan S1 Teknik Informatika ITS (2012-2016). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2013-2015) dan KMI (2013-2015). Penulis juga aktif dalam berbagai kegiatan kepanitiaan yaitu SCHEMATICS 2013 divisi dana usaha dan SCHEMATICS 2014 divisi dana usaha. Penulis juga meraih penghargaan *Seed For The Future* dari Huawei untuk mewakili ITS pada acara di Shenzhen dan Beijing. Penulis juga menyukai kegiatan sosial dan pecinta alam. Penulis memiliki hobi membaca buku dan menyukai hal baru. Penulis dapat dihubungi melalui email: yusufnugroho@hotmail.com.