TUGAS AKHIR – KI141502

DETEKSI COPY-MOVE UNTUK FORENSIK CITRA DENGAN MODIFIKASI ALGORITMA DUPLICATION DETECTION DAN ROBUST DETECTION

RAHMAT NAZALI SALIMI
NRP 5113100167

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

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

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

DETEKSI COPY-MOVE UNTUK FORENSIK CITRA DENGAN MODIFIKASI ALGORITMA DUPLICATION DETECTION DAN ROBUST DETECTION

RAHMAT NAZALI SALIMI
NRP 5113100167

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 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

COPY-MOVE DETECTION FOR IMAGE FORENSIC WITH MODIFICATION OF DUPLICATION DETECTION AND ROBUST DETECTION ALGORITHM

RAHMAT NAZALI SALIMI
NRP 5113100167

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

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

Department of Informatics
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

DETEKSI COPY-MOVE UNTUK FORENSIK CITRA DENGAN MODIFIKASI ALGORITMA DUPLICATION DETECTION DAN ROBUST DETECTION

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:
RAHMAT NAZALI SALIMI
NRP: 5113100167

Disetujui oleh Pembimbing Tugas Akhir:

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

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

SURABAYA
JANUARI, 2017

(Halaman ini sengaja dikosongkan)

DETEKSI *COPY-MOVE* UNTUK FORENSIK CITRA DENGAN MODIFIKASI ALGORITMA *DUPLICATION DETECTION* DAN *ROBUST DETECTION*

Nama Mahasiswa	**: RAHMAT NAZALI SALIMI**
NRP	**: 5113100167**
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 salah satu jenis serangan untuk memalsukan citra digital dimana pelaku menyalin beberapa daerah citra lalu meletakkannya pada bagian yang berbeda pada citra yang sama untuk menutupi bagian citra yang ada pada daerah tersebut. Oleh karena itu diperlukan metode untuk mendeteksi serangan copy-move untuk mengetahui tingkat keotentikan sebuah citra digital.

Data citra seringkali sengaja dilakukan proses seperti penambahan noise dan blurring agar serangan copy-move lebih sulit untuk dikenali, sehingga dibutuhkan tahapan praproses citra masukan berupa proses filtering untuk menghilangkan noise pada citra masukan. Tahapan pendeteksian serangan copy-move dapat dipersingkat dengan tidak melakukan tahap praproses namun menggunakan metode yang handal terhadap gangguan seperti noise dan blur. Oleh karena itu, dibutuhkan metode pendeteksian serangan copy-move yang handal terhadap berbagai variasi citra masukan sehingga tidak diperlukannya tahap praproses pada citra masukan.

Pada tugas akhir ini didapatkan sebuah metode yang dibuat dari modifikasi dua metode pendeteksian serangan copy-move yakni metode duplication detection yang efektif digunakan pada citra yang bebas gangguan dan metode robust detection yang efektif digunakan pada citra yang memiliki gangguan seperti noise

dan blur. Modifikasi metode dianggap efektif karena adanya toleransi antar fitur pada kedua metode yang digunakan. Saat citra masukan bebas gangguan maka fitur dari duplication detection akan menjadi dominan, sementara saat citra masukan memiliki gangguan seperti noise dan blur maka fitur dari robust detection yang akan menjadi dominan. Hal tersebut membuat metode dapat beradaptasi terhadap citra masukan sehingga tahap praproses citra tidak lagi diperlukan.

Kata kunci: Copy-Move, Metode duplication detection, Metode robust detection, Principal Component Analysis.

COPY-MOVE DETECTION FOR IMAGE FORENSIC WITH MODIFICATION OF DUPLICATION DETECTION ALGORITHM AND ROBUST DETECTION

Student's Name	**: RAHMAT NAZALI SALIMI**
Student's ID	**: 5113100167**
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 is a type of attack to falsify digital image where some region of the image are duplicated by copy and paste to another region within the same image to hide a part of the image in that region. Therefore a detection method is required to determine the authenticity level of the image.

A process like noise addition and blurring to the image make the copy-move attack harder to detect, therefore a preprocess is required to remove the noise from the image first then a detection method can be used. The copy-move attack detection stages can be shortened by not doing preprocess stage on the input image and replaced with a robust method that can detect copy-move attack even in a blurred and noised image. Therefore a robust copy-move attack detection method is required in order to pass through preprocess stage.

In this undergraduate thesis, a new method was obtained by modificating two copy-move attack detection method: duplication detection method that is effective to be used in a clean image and robust detection method that is effective to be used in a noisy or blurred image. Modificated method was considered effective because there is a tolerance between both method that have been used. When the input image are clean image then a feature from duplication detection will be a dominant feature, while when the input image are noisy or blurred image then a feature from robust

detection will be a dominant feature. It makes the method adaptive based on the input image, and therefore preprocess stage is no longer needed.

Keywords : Image Copy-Move, Duplication detection method, Robust detection method, Principal Component Analysis.

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, karena limpahan rahmat dari Allah Swt sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**DETEKSI COPY-MOVE UNTUK FORENSIK CITRA DENGAN MODIFIKASI ALGORITMA DUPLICATION DETECTION DAN ROBUST DETECTION**"

sebagai 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. Istri dan keluarga besar penulis yang selalu memberikan dukungan yang tak terhingga kepada penulis.
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 pembimbing II yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Bapak Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D. selaku dosen wali serta dosen penguji penulis yang telah memberikan arahan dan masukan kepada penulis.
5. Bapak Dr. Radityo Anggoro, S.Kom.,M.Sc. selaku koordinator mata kuliah Tugas Akhir yang memberikan banyak bantuan informasi.
6. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.

7. Teman-teman RMK KBJ dan KCV yang telah berjuang bersama penulis.
8. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, memberi ilmu dan berjuang bersama 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, Januari 2017

DAFTAR ISI

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kemajuan teknologi yang sangat pesat membuat manipulasi citra menjadi jauh lebih mudah. Salah satu jenis dari manipulasi citra adalah *copy-move* atau duplikasi sebagian daerah pada sebuah citra untuk diletakkan di lokasi lain guna menutupi sebagian objek yang tidak ingin ditampilkan. Telah banyak kasus penipuan yang melibatkan *copy-move* yang terjadi baik pada citra di koran maupun laporan resmi [1, 2]. Tidak hanya sekedar *copy-move* saja, seringkali pihak pemalsu citra melakukan serangkaian proses setelah manipulasi citra yang kemudian disebut dengan *post region duplication process* untuk mempersulit pihak lain dalam menemukan bukti kepalsuan gambar tersebut [3]. Beberapa contoh dari proses tersebut adalah *blurring*, *sharpening*, JPEG *Compression* [4] dan *noise addition* [5]. Sudah terdapat penelitian terkait algoritma deteksi *image forgery* untuk berbagai kasus seperti duplikasi sebuah daerah, *color filter interpolation*, dan *re-sampling* [6, 7, 8, 9]. Terdapat juga cara untuk mendeteksi *image forgery* dengan perhitungan langsung pada seluruh citra [10] maupun dengan memotong gambar menjadi beberapa bagian terlebih dahulu [11]. Setiap metode tersebut memiliki keunggulan masing-masing, namun memiliki kelemahan yang sama yakni pada pendeteksian citra yang telah dilakukan *post region duplication process* seperti penambahan *noise* dan operasi *blurring*.

Post region duplication process akan mengubah beberapa nilai piksel pada bagian yang dilakukan *copy-move*, sehingga menggunakan algoritma yang hanya membandingkan nilai piksel individu saja tidak cukup untuk mendeteksi daerah yang terduplikasi. Sebuah jurnal memberikan metode untuk mendapatkan karakteristik dari sebagian daerah gambar yang tidak akan berubah banyak meskipun dilakukan *blurring* dan

sharpening [3] sehingga handal untuk digunakan dalam membuktikan terjadinya *copy-move* pada citra yang telah dilakukan *post region duplication process*. Memanfaatkan sifat tersebut, diambil sebuah algoritma *duplication detecion* [7] yang memanfaatkan *principal component analysis* untuk digunakan sebagai kerangka awal dan kemudian dimodifikasi dengan melakukan penambahan beberapa metode berdasarkan pada algoritma *robust detection* [3].

Modifikasi dan implementasi ini diharapkan akan menghasilkan sebuah algoritma deteksi *copy-move* yang lebih cepat dari algoritma *brute force* biasa [7] dan juga handal untuk mendeteksi citra yang telah dilakukan *post region duplication process* [3].

1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana melakukan deteksi terhadap serangan *copy-move* pada citra yang telah dilakukan *post region duplication processing*?
2. Apakah metode modifikasi algoritma *duplication detection* dan *robust detection* dapat digunakan untuk mendeteksi serangan *copy-move* pada citra yang telah dilakukan proses *post region duplication procressing*?
3. Bagaimana cara untuk memodifikasi algoritma *duplication detection* dan *robust detection* agar handal dan dapat beradaptasi sesuai dengan jenis citra masukan?

1.3 Batasan Permasalahan

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

1. Bahasa pemrograman yang digunakan untuk implementasi deteksi serangan *copy-move* ini adalah *Python* versi 2.7.

2. Dataset yang digunakan adalah citra digital yang memiliki resolusi rendah berekstensi PNG.
3. Batasan *dataset* yang digunakan sebagai berikut:
    a. Daerah *copy-move* tidak berukuran terlalu kecil, dengan ukuran minimal 0.85% dari ukuran citra.
    b. Dataset tidak dilakukan proses *resize* dan *rotate*.
4. Keluaran dari implementesi ini adalah berkas citra digital dari *dataset* dengan kondisi sudah ditandai daerah yang diduga terdapat serangan *copy-move* menggunakan garis dan citra nol (hitam) dengan daerah terduga terduplikasi diberi warna putih.
5. Jumlah data yang digunakan adalah 30 citra yang dibuat sendiri dari 10 buah citra otentik dari *dataset*, serta dua buah citra dari *dataset* publik.
6. Metode yang digunakan untuk mendeteksi *copy-move* adalah metode *duplication detection*, metode *robust detection*, dan gabungan metode *duplication detection* dengan *robust detection.*
7. Perangkat lunak yang digunakan adalah *PyCharm* sebagai *IDE*, dan *Anaconda* sebagai kakas bantu.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah melakukan implementasi gabungan metode *duplication detection* dan *robust detection* untuk secara handal mendeteksi serangan *copy-move* pada citra digital baik serangan *copy-move* biasa maupun serangan *copy-move* yang telah dilakukan proses *post region duplication processing*.

1.5 Manfaat

Pembuatan tugas akhir ini diharapkan dapat memberikan manfaat pada dunia forensik digital untuk mendeteksi pemalsuan citra terhadap serangan *copy-move*, terutama pada kasus serangan *copy-move* yang telah dilakukan *post region duplication processing*, serta algoritma hasil modifikasi metode *duplication*

detection dan *robust detection* yang handal dan dapat beradaptasi terhadap berbagai jenis citra masukan.

Sedangkan bagi penulis, tugas akhir ini bermanfaat sebagai sarana untuk mengimplementasikan ilmu *image processing* 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 menganalisis dan mengidentifikasi pemalsuan citra terhadap serangan *copy-move* yang disertai dengan proses *post region duplication processing* menggunakan metode *duplication detection* dan *robust detection*.

1.6.2 Studi Literatur

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 *duplication detection*, *robust detection*, dan metode-metode *decomposition* seperti *principal component analysis*. Tugas akhir ini juga mengacu pada literatur jurnal karya Alin C. Popescu, dan Hany Farid dengan judul "*Exposing Digital Forgeries by Detecting Duplicated Image Region"* yang diterbitkan pada tahun 2004 [7], serta literatur jurnal karya Weiqi Luo, dan Jiwu Huang dengan judul "*Robust Detection of Region-Duplication Forgery in Digital Image"* yang diterbitkan pada tahun 2006 [3].

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 perangkat lunak yakni *PyCharm* sebagai *IDE*, dan *Anaconda* sebagai *interpreter* yang berisi bahasa *Python* versi 2.7, library berupa *Python Image Library* (*PIL*), *Numpy*, dan *Scipy*. Implementasi dilakukan pada sistem operasi *Microsoft Windows 10 64-bit*.

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 diujicoba dengan menggunakan perangkat lunak dengan tujuan mengetahui kemampuan metode yang digunakan dan mengevaluasi hasil tugas akhir dengan jurnal pendukung yang ada. Hasil evaluasi mencakup keakuratan dari hasil modifikasi, serta kendala yang dapat mempengaruhi kinerja metode tersebut dalam mendeteksi serangan *copy-move* yang disertai dengan proses *post region duplication*.

Beberapa alur pengujian adalah sebagai berikut:

1. Persiapan *dataset*
   Proses pengujian akan dilakukan dengan cara menyiapkan *dataset* citra otentik, kemudian dilakukan *copy-move*, dan kemudian dilakukan *copy-move* dengan disertai proses *post region duplication processing* dengan cara melakukan proses *blurring* pada daerah yang dilakukan *copy-move* yang dilakukan menggunakan aplikasi *Photoshop*. Citra dari *dataset* publik disiapkan dengan cara pengunduhan langsung dari laman web penyedia *dataset*.
2. Pengujian
   Pengujian dilakukan dengan melihat akurasi piksel daerah hasil deteksi dengan jawaban *dataset* yang sudah ada, serta

mengevaluasi hasil *confusion matrix* (*true positive*, *true negative*, *false positive*, dan *false negative*).

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup konsep, teori, implementasi, dan 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 teori forensik digital terutama pada citra digital, serangan *copy-move*, metode *duplication detection* dan metode *robust detection*.
3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi pembahasan mengenai perancangan dari metode *robust detection* yang mengusulkan cara perhitungan fitur citra dan *duplication detection* yang menggunakan *principal component analysis* yang digunakan untuk menedeteksi pemalsuan citra pada serangan *copy-move* yang telah dilakukan *post region duplication processing*.
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi modifikasi metode *duplication detection*, *robust detection*, dan modifikasi dari keduanya dalam bentuk *Pseudocode*.

5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dari modifikasi metode *duplication detection* dan *robust detection* yang digunakan untuk menedeteksi pemalsuan citra pada serangan *copy-move* yang sudah diimplementasikan pada *Pseudocode*. Uji coba dilakukan dengan menggunakan *dataset* citra yang memiliki resolusi rendah. Hasil evaluasi mencakup akurasi dari masing-masing metode yang dibuat untuk mendeteksi serangan *copy-move* pada citra digital.
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.

(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 *overlapping block* dari metode *duplication detection*, dan cara pencarian fitur karakteristik dari metode *robust detection*, dan beberapa teori lain yang mendukung pembuatan tugas akhir ini.

2.1 Forensik Digital

Forensik digital adalah aktivitas yang berhubungan dengan pemeliharaan, identifikasi, ekstraksi, dan dokumentasi bukti digital dalam kejahatan. Contoh bukti dalam forensik digital meliputi bukti fisik seperti komputer, dan bukti non-fisik seperti dokumen yang tersimpan dalam komputer (citra, video, berkas tulisan, rekaman lalu lintas data dalam jaringan). Karena luasnya cakupan, forensik digital dibagi menjadi beberapa cabang seperti forensik komputer, forensik analisis data, forensik jaringan, forensik perangkat bergerak, forensik basis data, dan lain lain [12].

2.2 Citra Digital

Citra digital adalah citra dua dimensi yang dihasilkan dari gambar *analog* dua dimensi yang kontinyu 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. Kemudian piksel tersebut direpresentasikan dengan barisnya dan kolomnya. Piksel [*x*, *y*] memiliki arti piksel yang terletak pada baris *x* dan kolom *y*.

2.3 Forensik Citra Digital

Forensik citra digital merupakan cabang dari keamanan multimedia yang bertujuan untuk melakukan penelusuran terkait

keaslian suatu citra dengan mengetahui informasi riwayatnya [13]. 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 ke bagian lainnya pada citra yang sama.

2.4 *Copy-Move* Citra

Copy-move adalah suatu metode penipuan citra digital dimana pemalsu menyalin sebagian citra lalu menggesernya ke bagian tertentu pada citra yang sama. Teknik ini biasanya digunakan untuk menutupi *point of interest* yang cenderung berupa objek dengan daerah citra digital yang diambil dari sekitarnya. Umumnya objek yang disalin adalah tekstur berwarna homogen seperti rerumputan, lautan, pemandangan langit, dan objek lain agar menutupi bagian citra ditempat lain yang memiliki warna serupa dengan objek tersebut agar sulit dibedakan dengan mata.

Gambar 2.1 Contoh Copy-Move Citra. (A) citra asli. (B) citra yang telah dilakukan *copy-move* pada daerah rumput untuk menutupi tempat sampah [7].

2.5 *Principal Component Analysis*

Principal Component Analysis (PCA) adalah salah satu metode analisis statistik multivariat yang digunakan untuk mereduksi jumlah dimensi data [14]. Proses reduksi dimensi dilakukan dengan melakukan transformasi data awal ke set variabel baru yang tidak berkorelasi satu sama lain, yang kemudian disebut dengan *principal component*. Berdasarkan derajat kepentingan, *principal component* akan terurut dari besar ke kecil dan hanya beberapa *principal component* teratas yang umumnya diambil untuk digunakan.

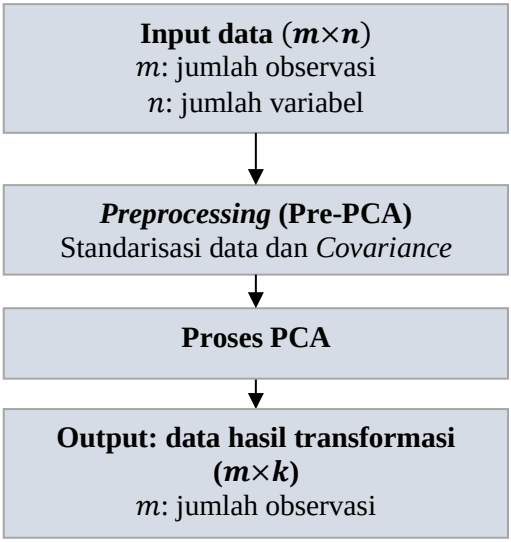
Penggunaan PCA menghasilkan data yang telah berkurang dimensinya dan hilangnya *multicollinearity*. Khusus dalam tugas akhir ini, maka PCA akan dimanfaatkan untuk mengurangi dimensi pada *array* yang merepresentasi blok citra. Setelah proses PCA, nilai *principal component* pada setiap blok akan digunakan untuk penentuan deteksi pemalsuan. Jika dalam beberapa blok terdapat nilai *principal component* yang mirip, maka diduga blok tersebut identik dan merupakan blok *copy-move*. Diagram alir proses *principal component analysis* dapat dilihat pada Gambar 2.2.

2.6 *Overlapping Block*

Overlapping block merupakan kumpulan blok citra berukuran $L \times L$ yang didapat dari dekomposisi sebuah citra berukuran $M \times N$. Blok citra tersebut saling bertumpukan antar satu sama lain dengan selisih koordinat sebesar satu piksel. Jumlah blok citra yang saling *overlap* (N_b) pada sebuah citra dapat dihitung menggunakan Persamaan 2.1, dimana i, j=(1, 2, 3, …) adalah jumlah lompatan iterasi yang dilakukan saat pembuatan *overlapping block*. Nilai satu pada *i* dan *j* memiliki arti bahwa pembuatan *overlapping block* dilakukan sesuai teori dengan iterasi pergeseran sejumlah satu piksel. Nilai *a* pada *i* dan *j* memiliki arti bahwa pembuatan *overlapping block* dilakukan dengan iterasi pergeseran sejumlah *a* piksel. Semakin besar nilai *i* dan *j*, maka

metode akan makin cepat selesai karena jumlah *overlapping block* berkurang sebanyak $\frac{1}{i+j}\times 100\%$. Contoh *overlapping block* pada matriks dengan lebar 5x5 terdapat pada Gambar 2.3

$$N_b = \left(\frac{M-L+1}{i}\right)\times\left(\frac{N-L+1}{j}\right) \tag{2.1}$$

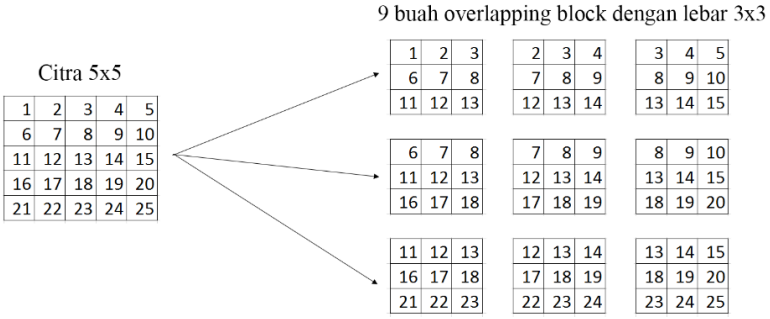


Gambar 2.2 Diagram blok dari *principal component analysis*

Setiap *overlapping block* memiliki koordinat yang merepresentasikan tempat *overlapping block* tersebut berada. Terdapat banyak aturan untuk mengambil koordinat tersebut. Pada kasus ini, koordinat tersebut diambil dari ujung kiri atas dari *overlapping block*.

2.7 *Lexicographical Sorting*

Lexicographical sorting merupakan algoritma *sorting* untuk mengurutkan serangkaian data agar menjadi susunan sesuai dengan urutan abjad [15]. Penggunaan metode *sorting* ini sering dijumpai pada pembuatan kamus literatur. Contoh kasusnya adalah dimana huruf *x* akan lebih dulu ditemukan sebelum huruf *y*, serta *a* akan lebih dahulu ditemukan dari pada *aa*.



Gambar 2.3 Contoh *overlapping block* pada matriks dengan lebar 5x5

2.8 Metode *Duplication Detection*

Metode *duplication detection* [7] mengusulkan penggunaan *principal component analysis* untuk menghitung fitur dari blok citra. Dikatakan bahwa metode ini lebih cepat dalam segi komputasi dibandingkan dengan algoritma deteksi serangan *copy-move* yang lain. Namun algoritma ini akan turun akurasinya saat mendapati citra masukan dengan kualitas rendah atau citra yang memiliki banyak *noise*.

Metode ini dimulai dengan membagi citra berukuran $M \times N$ menjadi blok citra dengan ukuran $L \times L$ yang relatif kecil dan saling bertumpuk satu dengan yang lainnya. Blok citra tersebut kemudian disebut juga dengan *overlapping block*. Jumlah *overlapping block*

pada sebuah citra, yang kemudian diberi nama variabel N_b dapat dihitung menggunakan Persamaan 2.1.

Setelah kumpulan *overlapping block* dari sebuah citra didapatkan, akan dilakukan ekstraksi fitur untuk mendapatkan fitur dari setiap blok citra. Pada tahap ini digunakan *principal component analysis*. Seluruh kumpulan blok citra yang berisi sejumlah b piksel akan diproses sebagai berikut.

Tahap pertama, persiapkan *array* masukan $\vec{x}$. *Array* ini berisi nilai piksel dari blok citra yang dibentuk vektor dan memiliki sejumlah N_b elemen. Untuk citra *grayscale*, maka *array* masukan merupakan *array* berukuran $M \times N$ yang berisi nilai piksel pada koordinat yang bersesuaian yang kemudian dibentuk vektor sehinggal menjadi 1 baris. Untuk citra berwarna, terdapat dua opsi. Opsi pertama, hitung PCA pada setiap *color channel* secara terpisah. Opsi kedua, bangun blok piksel dengan ukuran $3b$ piksel yang tediri dari masing-masing *color channel*, kemudian lakukan proses PCA.

Tahap kedua, setelah pembuatan *array* masukan selesai, tentukan nilai jumlah dimensi N_t agar memenuhi Persamaan 2.2,

$$1 - \epsilon = \frac{\sum_{i=1}^{N_t} \lambda_i}{\sum_{i=1}^{b} \lambda_i} \tag{2.2}$$

dimana ϵ adalah pecahan desimal varian yang diabaikan, b adalah jumlah piksel dari sebuah blok citra, dan $\lambda_i$adalah eigenvalue pada blok citra ke i. Tahap kedua ini memiliki beberapa langkah yakni menghitung matriks kovarian dari *array* $\vec{x}$ dengan Persamaan 2.3,

$$C = \sum_{i=1}^{N_b} \overrightarrow{x_i}\, \overrightarrow{x_i}^{\,T} \tag{2.3}$$

kemudian menghitung *eigenvector* $\overrightarrow{e_j}$ dan *eigenvalue* λ_j sehingga memenuhi Persamaan 2.4.

$$C\vec{e_j} = \lambda_j \vec{e_j} \tag{2.4}$$

Menggunakan *eigenvector* $\vec{e_j}$, dapat ditentukan *principal component* dengan membentuk matriks 1 dimensi yang baru untuk setiap blok citra $\vec{x_i}$ menggunakan Persamaan 2.5,

$$\vec{x_i} = \sum_{j=1}^{b} a_j \vec{e_j} \tag{2.5}$$

dengan $j = 1 \dots, b$ dan $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_b$, dimana $a_j = \vec{x_i}^T \vec{e_j}$ dan $\vec{a_i} = (a_1 \dots a_1)$ merupakan *principal component* yang dapat digunakan sebagai representasi data yang baru.

Setelah tahap kedua selesai, maka akan didapatkan sebuah kontainer $\vec{s}$ berisi kumpulan pasangan data berupa koordinat blok citra yang direpresentasikan dengan koordinat ujung kiri atas dari blok citra tersebut, serta hasil *principal component* yang sudah didapat. Contoh data keluaran tersebut dapat dilihat pada Gambar 2.4.

```
[(0, 0), [0.99452807710343727, 0.00072308833100219331,
[(0, 1), [0.99442599058010706, 0.00071483057717183914,
[(0, 2), [0.99424857203982842, 0.00071454435151663482,
[(0, 3), [0.99413887619756902, 0.00072667324640356492,
[(0, 4), [0.99391929193715634, 0.00074675808099707833,
[(0, 5), [0.99386293336933906, 0.00079088481957302701,
[(0, 6), [0.99366037117549344, 0.00087951390395742677,
[(0, 7), [0.9934739931733193, 0.00088233385951487161,
[(0, 8), [0.99322226758263354, 0.00088019576111375997,
[(0, 9), [0.99303073893028226, 0.00088684900761483237,
```

Gambar 2.4 Contoh potongan hasil pencarian fitur dari 10 *overlapping block* dengan *principal component analysis*

Tahap ketiga, kontainer $\vec{s}$ tersebut kemudian di urutkan menggunakan metode *lexicographical sorting* berdasarkan nilai *principal component* setiap blok citra. Setelah pengurutan, maka

blok citra dengan *principal component* yang sama atau bermiripan akan terletak berdekatan.

Tahap keempat, buat sebuah matriks baru $\vec{t}$ dan mengisinya dengan pasangan koordinat blok citra (x_i, y_i) dan (x_j, y_j) hanya jika $|i - j| < N_n$, dimana i dan j merupakan indeks blok citra yang berada pada kontainer $\vec{s}$ dan N_n merupakan batas maksimal jarak tetangga yang diperhitungkan.

Tahap kelima, hitung *offset* dari setiap elemen matriks $\vec{t}$ tersebut dengan Persamaan 2.6. Tahap keenam, buang semua pasangan koordinat pada matriks $\vec{t}$ yang memiliki frekuensi *offset* kurang dari *Nf*.

$$offset = \begin{cases} (x_i - x_j, y_i - y_j), \text{jika } x_i - x_j > 0 \\ (x_j - x_i, y_i - y_j), jika\ x_i - x_j < 0 \\ (0, |y_i - y_j|), jika\ x_i = x_j \end{cases} \quad (2.6)$$

Tahap ketujuh, buang semua pasangan koordinat pada matriks $\vec{t}$ yang memiliki besar *offset* kurang dari N_d, dimana besar *offset* dihitung dengan Persamaan 2.7.

$$N_d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (2.7)$$

Tahap terakhir, buat citra *ground truth* dengan membuat citra biner (hitam putih) dengan nilai bawaan 0, dan beri nilai 1 pada setiap lokasi pasangan blok citra yang tersisa. Dari citra *ground truth* tersebut, duplikat citra masukan dan beri garis berwarna pada setiap tepi dari daerah putih pada *ground truth* untuk mendapatkan citra hasil deteksi.

2.9 *Metode Robust Detection*

Metode *robust detection* [3] mengusulkan penggunaan aturan perbandingan nilai piksel untuk menghitung fitur dari blok

citra. Dikatakan bahwa metode ini lebih lambat dalam segi komputasi dibandingkan dengan algoritma *duplication detection*. Namun algoritma ini bersifat handal pada berbagai citra masukan karena aturan perbandingan nilai piksel yang diajukan akan cenderung tetap pada kasus citra yang dilakukan proses *post region duplication process* seperti proses *blurring* dan penambahan *noise*.

Metode ini dimulai dengan membagi citra berukuran $M \times N$ menjadi blok citra dengan ukuran $L \times L$ yang relatif kecil dan saling bertumpuk satu dengan yang lainnya. Blok citra tersebut kemudian disebut juga dengan *overlapping block*. Jumlah *overlapping block* pada sebuah citra, yang kemudian diberi nama variabel N_b dapat dihitung menggunakan Persamaan 2.1.

Setelah kumpulan *overlapping block* dari sebuah citra didapatkan, akan dilakukan ekstraksi fitur untuk mendapatkan fitur dari setiap blok citra. Pada tahap ini digunakan aturan perbandingan nilai piksel untuk menghitung fitur dari blok citra. Seluruh kumpulan blok citra yang berisi sejumlah b piksel akan diproses sebagai berikut:

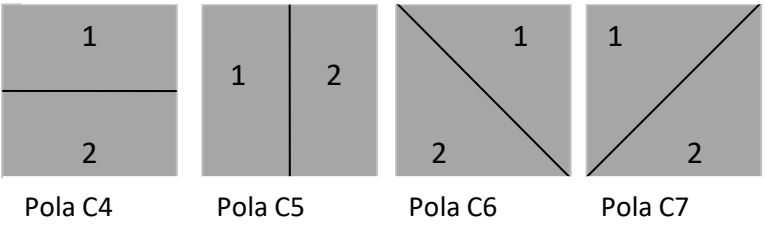
Tahap pertama, jika citra masukan memiliki *colorpsace* selain *grayscale*, maka rubah *colorspace* citra tersebut menjadi RGB.

Tahap kedua, hitung tiga karakteristik pertama (c_1, c_2, c_3). c_1 merupakan fitur rataan jumlah total piksel merah, c_2 merupakan fitur rataan jumlah total piksel hijau, c_3 merupakan fitur rataan jumlah total piksel biru. Jika citra masukan memiliki *colorspace graycale*, maka nilai c_1, c_2, dan c_3 adalah nol.

Tahap ketiga, rubah *colorspace* blok citra menjadi *Y channel* dengan mengganti nilai setiap piksel dengan nilai baru yang didapat dari Persamaan 2.8, dimana R, G, dan B adalah nilai piksel merah, hijau, dan biru pada sebuah piksel.

$$Y = 0.299\,R + 0.587G + 0.114B \tag{2.8}$$

Tahap keempat, bagi blok citra tersebut kedalam empat macam pola yang saling membagi blok citra menjadi dua bagian yang sama seperti pada Gambar 2.5.



Gambar 2.5 Pola daerah pembagian blok citra

Tahap kelima, hitung empat karakteristik selanjutnya (c_4, c_5 , c_6 , c_7). Masing-masing karakteristik dihitung dengan Persamaan 2.9 sesuai pada masing-masing pola yang diberikan pada Gambar 2.5. Setiap hasil perhitungan tersebut kemudian disimpan pada matriks $\vec{s}$ beserta dengan koordinat masing-masing blok citra. Contoh hasil penghitungan fitur karakteristik terdapat pada Gambar 2.6.

$$c_i = \frac{sum\big(part1(i)\big)}{sum\big(part1(i) + part2(i)\big)}, i = 4{,}5{,}6{,}7 \qquad (2.9)$$

Tahap keenam, lakukan pengurutan menggunakan metode *lexicographical sorting* pada matriks $\vec{s}$ berdasarkan nilai fitur karakteristiknya. Hasil pengurutan akan mendekatkan blok citra yang memiliki fitur karakteristik yang sama atau berdekatan.

Tahap ketujuh, buat sebuah matriks baru $\vec{t}$ dan mengisinya dengan pasangan koordinat blok citra (x_i, y_i) dan $\big(x_j, y_j\big)$, dimana i dan j merupakan indeks blok citra yang berada pada kontainer $\vec{s}$.

Tahap kedelapan, menghitung perbedaan nilai fitur menggunakan Persamaan 2.10 dimana i dan j merupakan pasangan koordinat blok citra dan k merupakan indeks fitur

karakteristik, buang semua pasangan koordinat pada matriks $\vec{t}$ yang tidak memenuhi Persamaan 2.11, dimana $P(k)$, t_1, dan t_2 berlaku sebagai *threshold*.

```
[(0, 0), [192, 199, 212, 0.5318353171658404,
[(0, 1), [192, 199, 212, 0.5318431000019791,
[(0, 2), [192, 199, 212, 0.5318195088399895,
[(0, 3), [192, 199, 212, 0.5318667722458675,
[(0, 4), [192, 199, 211, 0.5318843520915522,
[(0, 5), [192, 199, 211, 0.5319141562909973,
[(0, 6), [192, 199, 211, 0.5319442862730923,
[(0, 7), [192, 199, 211, 0.5319800121828618,
[(0, 8), [192, 199, 211, 0.5319804446998103,
[(0, 9), [192, 199, 211, 0.5319840463745138,
```

Gambar 2.6 Contoh potongan hasil pencarian fitur dari 10 *overlapping block* dengan aturan perbandingan nilai piksel

Tahap kesembilan, buat *histogram* dari kumpulan *offset* tersebut dan ambil *offset* yang paling banyak frekuensinya. Buang pasangan koordinat yang memiliki *offset* selain *offset* tersebut.

$$Diff(k) = |c_k(i) - c_k(j)| \tag{2.10}$$

$$\begin{gathered} Diff(k) < P(k) \\ Diff(1) + Diff(2) + Diff(3) < t_1 \\ Diff(4) + Diff(5) + Diff(6) + Diff(7) < t_2 \end{gathered} \tag{2.11}$$

Terakhir, dari pasangan koordinat blok citra yang tersisa, buat citra *ground truth* dengan membuat citra biner (hitam putih) dengan nilai awal 0, dan beri nilai 1 pada setiap lokasi pasangan blok citra. Dari citra *ground truth* tersebut, duplikat citra masukan dan beri garis berwarna pada setiap tepi dari daerah putih pada *ground truth* untuk mendapatkan citra hasil deteksi.

2.10 *Post Region Duplication Process*

Post region duplication process adalah serangkaian proses yang dilakukan pada citra setelah dilakukan serangan *copy-move* dengan tujuan lebih menutupi blok yang diduplikasi agar citra yang dipalsukan lebih sulit untuk dideteksi. Beberapa contoh proses tersebut adalah seperti proses *blurring* dan *sharpening* pada citra yang dilakukan pada tepi blok citra yang diduplikasi untuk menghaluskan batas antar bagian citra yang otentik dengan bagian citra yang dipalsukan [3].

2.11 *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" dapat digunakan untuk *create, export, list, remove* dan *update* lingkungan yang bisa digunakan untuk versi bahasa pemrograman *Python* atau paket yang dipasang didalamnya [16].

2.12 *OpenCV*

*Open*CV (*Open Source Computer Vision*) adalah *library* yang utamanya digunakan untuk pemrosesan citra 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* [17].

2.13 *NumPy*

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

1. Objek *array N* dimensi.
2. Kakas bantu untuk mengintegrasi dengan *Pseudocode C/C++* dan *Fortran*.
3. Memiliki fungsi yang banyak digunakan pada aljabar linier dan operasi matriks, menjadikannya mudah digunakan untuk kasus tertentu seperti penjumlahan dan perkaian matriks, pencarian *histogram*, dan *principal component analysis*.

Selain digunakan untuk hal ilmiah, *NumPy* juga bisa digunakan untuk *container* multidimensi yang efisien untuk data yang umum digunakan. 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*, yakni sebuah modul *Open Source* untuk bahasa pemrograman *Python* yang menyediakan fungsi-fungsi yang telah siap digunakan (*precompiled). NumPy* memperkaya kemampuan bahasa pemrograman *Python* dengan data struktur yang efisien untuk komputasi yang membutuhkan *arrays* atau *matrices* multidimensi [18].

2.14 *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* [19].

2.15 *Mean Squared Error*

Mean Squared Error adalah metode untuk mengevaluasi kemiripan dari dua buah matriks atau *array* dengan cara menjumlahkan selisih atau perbedaan keduanya yang kemudian dikuadratkan. Pendekatan ini menghasilkan nilai kesalahan peramalan yang cenderung besar karena kesalahan-kesalahan itu dikuadratkan sebelum dijumlahkan. Nilai MSE memiliki rentang nol hingga tak terhingga, dan semakin kecil nilai MSE maka semakin mirip kedua buah matriks yang dibandingkan [20]. Kode fungsi MSE terdapat pada *Pseudocode* 2.1.

```
1   def mse(imageA, imageB):
2       err = np.sum((
            imageA.astype("float") -
            imageB.astype("float")) ** 2)
3       err /= float(imageA.shape[0] * imageA.shape[1])
4       return err
```

Pseudocode* 2.1 Fungsi MSE dengan bahasa *python

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 gabungan metode *duplication detection* dan *robust detection.*

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 awal yang akan dimasukkan ke dalam sistem. Data yang digunakan dalam perangkat lunak untuk mendeteksi serangan *copy-move* pada citra digital terdiri dari 30 buah data masukan yang dibuat sendiri. Data masukan tersebut adalah 10 buah citra otentik, 10 buah citra sama yang mengalami serangan *copy-move*, dan 10 buah citra sama yang mengalami serangan *copy-move* disertai proses *blurring*, ditambah dengan dua buah citra dari *dataset* publik. Proses *blurring* tersebut berlaku sebagai *post region duplication processing*. Semua data masukan diujikan pada algoritma *duplication detection*, algoritma *robust detection*, dan modifikasi kedua algoritma tersebut yang akan dijelaskan selanjutnya.

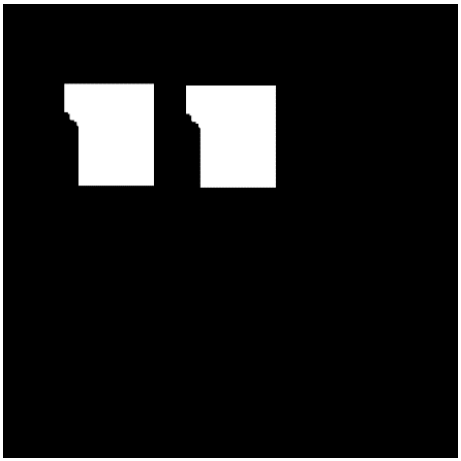
Setiap citra *dataset* memiliki pasangan citra yang berlaku sebagai *ground truth*, menggambarkan daerah mana saja dari citra yang terkena serangan *copy-move*. Hasil proses deteksi kemudian akan dicocokkan dengan pasangan data *ground truth* dengan

menggunakan *mean squarred error* dan *similarity* untuk mendapatkan tingkat akurasi proses deteksi.

Contoh citra otentik, citra palsu, dan citra *ground truth* sebagai data masukan ditunjukkan pada Gambar 3.1, Gambar 3.2, dan Gambar 3.3 yang mencoba menutup patung burung yang terdapat pada bagian tengah citra tersebut.

Gambar 3.1 Contoh data masukan citra yang otentik

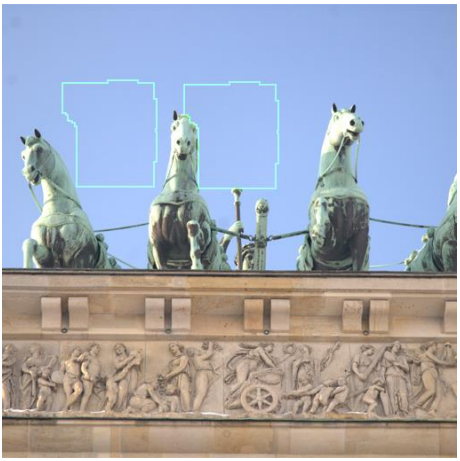
Gambar 3.2 Contoh data masukan citra yang terkena serangan *copy-move*



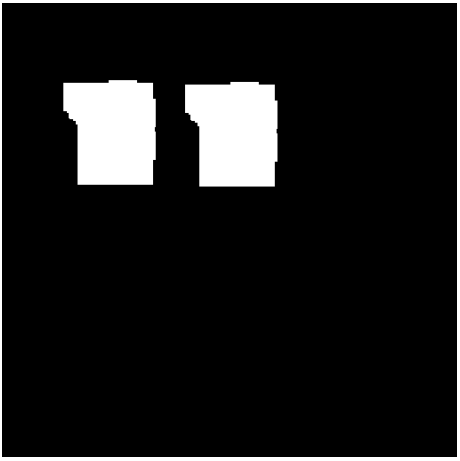
Gambar 3.3 Contoh data *ground truth* dari hasil deteksi citra yang terkena serangan *copy-move*

3.1.2 Data Keluaran

Data masukan akan diproses dengan menggunakan metode *duplication detection*, *robust detection*, dan modifikasi keduanya. Hasil dari proses deteksi serangan *copy-move* pada masing-masing metode tersebut adalah dua citra yakni citra masukan dengan daerah yang diduga dipalsukan yang telah ditandai dengan garis berwarna dan citra hasil deteksi dengan bawaan berwarna hitam sementara daerah yang diduga diduplikasi diberi warna putih, yang ditunjukkan pada Gambar 3.4 dan Gambar 3.5. Untuk pengukuran akurasi, maka dibuat program lain yang berfungsi untuk menghitung kemiripan hasil keluaran dengan *ground truth* dengan menggunakan metode *mean squarred error* dan *similarity*.



Gambar 3.4 Contoh citra hasil keluaran yang memberi garis warna pada daerah yang diduga dipalsu



Gambar 3.5 Contoh citra hasil keluaran yang menunjukkan hasil deteksi pemalsuan

3.2 Desain Umum Sistem

Rancangan perangkat lunak pendeteksi serangan *copy-move* pada citra yang dibuat melibatkan dua buah metode, yakni *duplication detection* [7] dan *robust detection* [3]. Kedua metode tersebut memiliki persamaan konsep yakni dengan membagi citra berukuran $M \times N$ menjadi blok citra dengan ukuran $L \times L$ yang relatif kecil dan saling bertumpuk satu dengan yang lainnya. Blok citra tersebut kemudian disebut juga dengan *overlapping block*, yang jumlahnya dapat dihitung menggunakan Persamaan 3.1, dimana i, j=(1, 2, 3, …) adalah jumlah lompatan iterasi yang dilakukan saat pembuatan *overlapping block*. Nilai satu pada *i* dan *j* memiliki arti bahwa pembuatan *overlapping block* dilakukan sesuai teori dengan iterasi pergeseran sejumlah satu piksel. Nilai *a* pada *i* dan *j* memiliki arti bahwa pembuatan *overlapping block* dilakukan dengan iterasi pergeseran sejumlah *a* piksel. Semakin besar nilai *i* dan *j*, maka metode akan makin cepat selesai karena jumlah *overlapping block* berkurang sebanyak $\frac{1}{i+j} \times 100\%$. Berkurangnya jumlah *overlapping block* berarti menurunnya akurasi deteksi karena metode sangat bergantung pada atribut dan jumlah *overlapping block*. Pada tugas akhir ini, digunakan nilai *i* dan *j* sebesar satu satuan untuk mendapatkan akurasi terbaik dengan waktu jalannya program yang lama sebagai *trade-off*.

Setelah kumpulan *overlapping block* dari sebuah citra didapatkan, akan dilakukan ekstraksi fitur untuk mendapatkan fitur dari setiap blok citra. Kedua metode tersebut memiliki perbedaan pada cara mendapatkan fitur dari blok citra. Metode *duplication detection* mengambil fitur dengan memanfaatkan salah satu proses dekomposisi yakni *principal component analysis*. Sementara metode *robust detection* mengusulkan aturan sendiri untuk menghitung fitur dengan cara perbandingan nilai piksel pada blok citra. Kumpulan fitur tersebut kemudian diurutkan berdasarkan nilai fitur yang didapat, dengan harapan bahwa citra blok yang identik akan memiliki nilai fitur yang cenderung mirip pula, dan

akan terletak saling berdekatan pada hasil *sorting*. Diagram alir dari desain umum ini ditunjukkan pada Gambar 3.6.

$$N_b = \frac{(M - L + 1)}{i} \times \frac{(N - L + 1)}{j} \tag{3.1}$$

Metode pertama yakni metode *duplication detection* mengusulkan penggunaan *principal component analysis* untuk menghitung fitur dari blok citra yang telah dijelaskan pada Bab 2.8. Metode pertama tersebut cepat, namun tidak handal dalam kasus citra masukan memiliki banyak *noise*, atau jika daerah yang diduplikasi dilakukan proses *blurring*. Metode ini sudah memiliki asumsi bahwa daerah yang terduplikasi dapat lebih dari satu pasang, sehingga metode akan dapat mendeteksi citra dengan daerah duplikasi lebih dari satu pasang. Metode ini menggunakan *offset* sebagai representasi pergerakan sebuah blok citra. *Offset* merupaka data *tuple* yang berisi dua buah elemen yang merepresentasikan jarak pasangan blok citra yang identik pada sumbu *x* dan sumbu *y* (*x, y*). Metode ini melakukan operasi *absolut* untuk mendapatkan nilai mutlak pada penghitungan *offset* sehingga blok yang bergerak berlawanan arah dengan besar sama akan dianggap memiliki *offset* yang sama. Toleransi pada nilai *offset* berupa proses *absolut* tersebut justru akan meningkatkan kemungkinan blok-blok yang kebetulan memiliki *offset* yang sama untuk ikut terduga sebagai hasil duplikasi, padahal belum tentu blok-blok tersebut benar-benar hasil serangan *copy-move*. Diagram alir dari metode *duplication detection* ditunjukkan pada Gambar 3.7.

Metode kedua yakni metode *robust detection*, mengusulkan aturan penghitungan fitur dari blok citra yang telah dijelaskan pada Bab 2.9. Metode tersebut lambat karena menghitug nilai perbandingan piksel dari setiap blok citra. Perhitungan tersebut melibatkan seluruh nilai piksel yang ada pada blok citra tersebut dan dilakukan secara berkali-kali. Tiga karakteristik pertama menghitung rataan dari nilai piksel, sehingga akan dilakukan tiga

kali iterasi sebanyak piksel pada blok citra untuk menghitung jumlah nilai piksel dari *colorspace* merah, hijau, dan biru. Pada empat karakteristik selanjutnya diperlukan empat kali iterasi sebanyak piksel pada blok citra untuk mendapatkan empat buah fitur karakteristik.

Metode ini memanfaatkan *histogram* untuk pengukuran frekuensi *offset*, lalu hanya mengambil *offset* dengan frekuensi tertinggi. Hal ini meangkibatkan metode ini hanya bisa mendeteksi satu daerah duplikasi saja karena hanya *offset* dengan frekuensi tertinggi saja yang digunakan.

Di sisi lain, metode ini handal dalam menangani berbagai variasi citra masukan. Telah dibuktikan bahwa meski citra masukan memiliki kualitas rendah seperti memiliki banyak *noise* dan telah dilakukan proses *blur*ing, tidak dilakukannya *preprocessing* seperti *noise reduction* maupun *smoothing* akan tetap menghasilkan jawaban yang baik. Hal ini dikarenakan fitur karakteristik yang diusulkan handal terhadap operasi citra. Pada contoh kasus proses *blurring*, maka proses perataan nilai piksel ini tidak mengurangi jumlah total piksel karena *blurring* hanya meratakan nilai piksel tanpa ada pengurangan nilai piksel. Karena total nilai piksel pada sebuah blok citra tersebut tidak berubah, maka fitur karakteristikpun tidak akan jauh berubah karena didapat dari nilai perbaindangan piksel, sehingga dapat dikatakan handal. Diagram alir dari metode *robust detection* ditunjukkan pada Gambar 3.8.

Masing-masing dari kelemahan kedua metode tersebut kemudian dicoba untuk diperbaiki. Pada metode pertama, persamaan untuk mendapatkan *offset* diganti yang semula menggunakan operasi *absolut* kemudian dihapus sehingga hasil perhitungan *offset* akan memiliki tanda negatif dan positif. Keadaan ini akan membuat representasi *offset* lebih akurat karena tidak bercampurnya *offset* negatif dan positif yang bernilai sama. Pada metode kedua yang semula hanya dapat mendeteksi satu daerah duplikasi saja karena hanya mengambil frekuensi *offset* tertinggi, kemudian diberi toleransi agar dapat mengambil semua

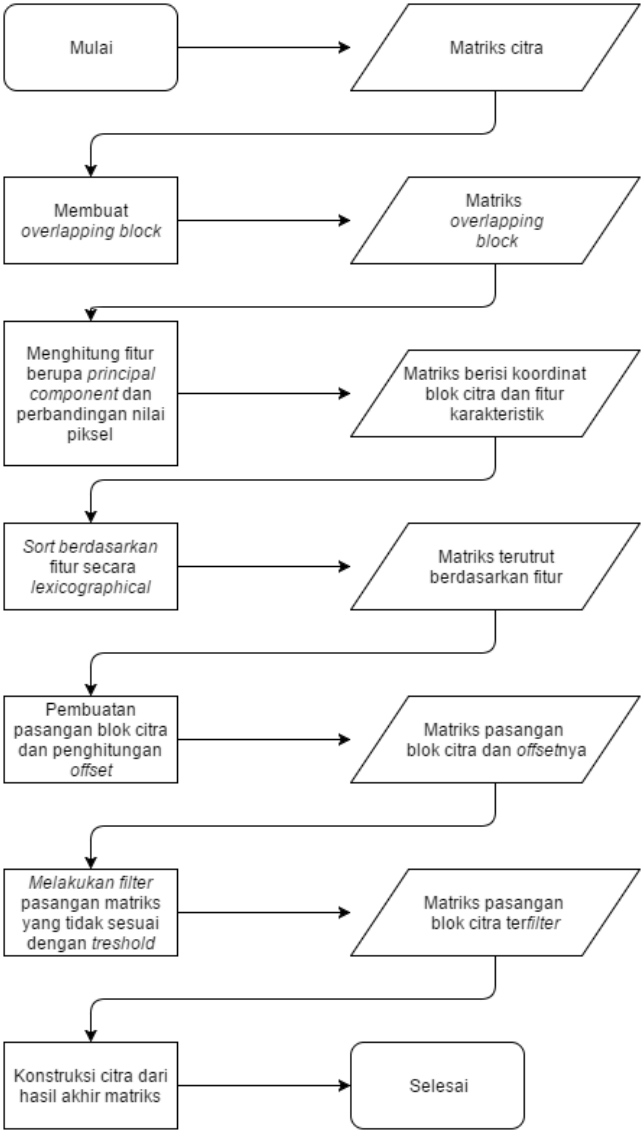
offset yang memiliki frekuensi *offset* melebihi *threshold* frekuensi *offset* minimal.

Menggunakan kedua metode yang telah diperbaiki tersebut, dibuat sebuah algoritma baru yang menggunakan masing-masing metode yang diusulkan dan diberikan teknik toleransi agar sebuah metode tidak dominan terhadap metode lainnya. Teknik toleransi tersebut dapat menggunakan operasi *round* atau pembulatan pada hasil fitur karakteristik agar semua fitur karakteristik dapat ikut andil dalam mendekatkan pasangan blok citra yang diduga identik pada saat pengurutan fitur karakteristik.

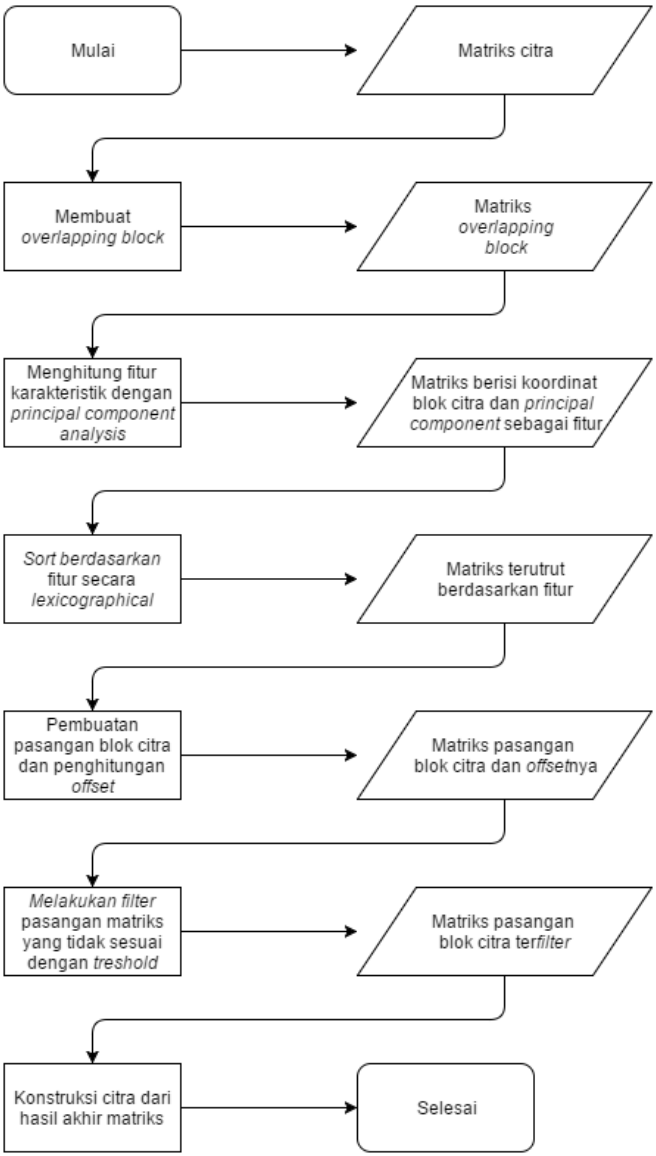
Rancangan perangkat lunak pendeteksi serangan *copy-move* pada citra menggunakan modifikasi metode *duplication detection* dan *robust detection* dimulai dengan menghitung masing-masing fitur karakteristik setiap metode dan kemudian dimasukkan kedalam satu kontainer sekaligus, sehingga sebuah elemen pada kontainer memiliki tiga data yakni: koordinat blok, *principal component* yang merupakan penerapan dari metode pertama, dan fitur karakteristik yang merupakan penerapan dari metode kedua. Masing-masing nilai karakteristik tersebut dilakukan proses *round* menuju tingkat ketelitian tertentu untuk menghapus beberapa digit desimal. Pembulatan ini sekilas akan menurunkan akurasi dari masing-masing metode, namun akan menimbulkan toleransi antar metode yang mengakibatkan hilangnya dominasi salah satu metode pada proses pengurutan nantinya.

Kontainer tersebut kemudian diurutkan dengan metode *lexicographical sorting* dengan berdasarkan pada nilai karakteristik yang didapat dari kedua metode. Proses pengurutan ini akan mendekatkan blok-blok citra yang diduga indentik karena memiliki nilai fitur karakteristik yang sama baik dari metode pertama dan metode kedua.

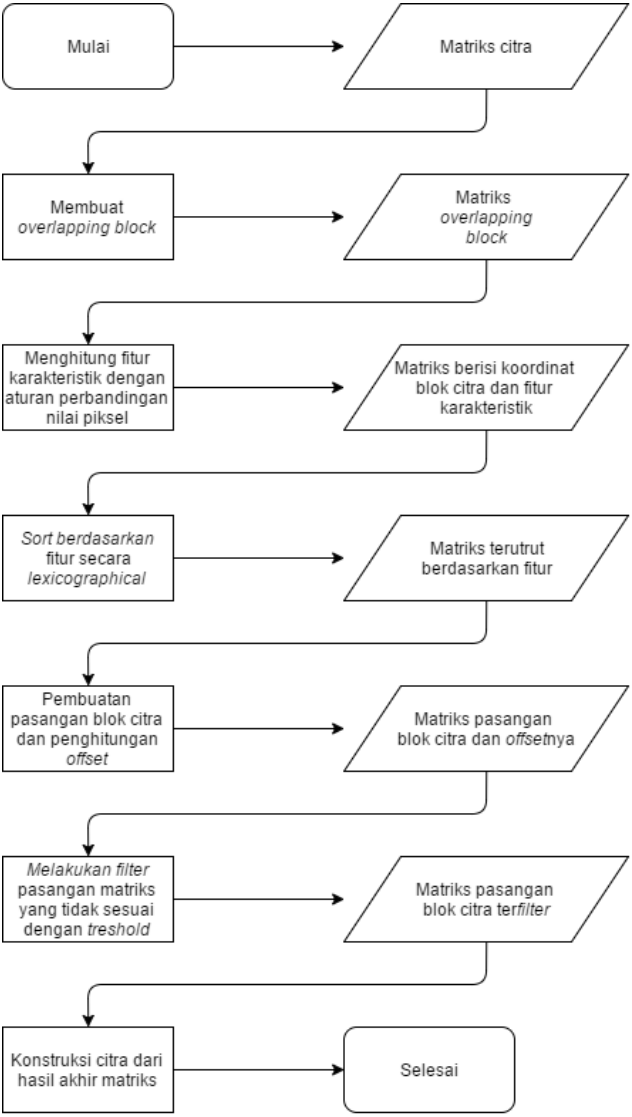
Dari hasil pengurutan tersebut kemudian dibuat pasangan blok citra yang saling berdekatan dengan *threshold* tertentu, dengan asumsi bahwa blok yang identik hanya akan terletak saling berdekatan.



Gambar 3.6 Diagram alir metode hasil modifikasi sebagai desain umum



Gambar 3.7 Diagram metode proses *duplication detection*



Gambar 3.8 Diagram alir metode *robust detection*

Hasil pemasangan tersebut kemudian dilakukan *filter* dengan menghapus pasangan blok citra yang tidak memenuhi syarat-syarat tertentu. Proses *filter* tersebut akan menyisakan pasangan blok citra yang memang benar-benar diduga identik. Kemudian dibuatlah citra jawaban berdasarkan pasangan koordinat blok citra yang tersisa.

3.3 Modifikasi *Duplicaton Detection* dan *Robust Detection*

Seperti yang telah dijelaskan pada Bab 3.2, bahwa dibuat algoritma baru yang menggabungkan kedua metode. Metode pertama adalah *duplication detection*, dan metode yang kedua adalah *robust detection*. Metode pertama berjalan dengan cepat, namun sensitif terhadap citra kualitas rendah seperti yang memiliki banyak *noise*. Operasi *blurring* pada citra yang telah dilakukan serangan duplikasi dapat langsung menggagalkan proses deteksi. Metode kedua berjalan lambat, namun handal terhadap variasi citra dan dapat mengenali daerah yang sama meski telah dilakukan proses *blurring*.

Metode gabungan ini kemudian diberi nama metode *robust-duplication detection*, dimulai dengan membagi citra berukuran $M \times N$ menjadi blok citra dengan ukuran $L \times L$ yang relatif kecil dan saling bertumpuk satu dengan yang lainnya. Blok citra tersebut kemudian disebut juga dengan *overlapping block*. Jumlah *overlapping block* pada sebuah citra, yang kemudian diberi nama variabel N_b dapat dihitung menggunakan Persamaan 3.1.

Setelah kumpulan *overlapping block* dari sebuah citra didapatkan, dilakukan ekstraksi fitur untuk mendapatkan fitur dari setiap blok citra. Pada tahap ini terdapat dua proses untuk menghitung fitur. Pertama adalah *principal component analysis*, dan kedua adalah aturan perbandingan nilai piksel.

Pada tahap pertama proses pertama yakni *principal component analysis*, seluruh kumpulan blok citra yang berisi sejumlah b piksel akan diproses sebagai berikut.

Persiapkan *array* masukan $\vec{x}$. *Array* ini berisi nilai piksel dari blok citra yang dibentuk vektor dan memiliki sejumlah N_b elemen. Untuk citra *grayscale*, maka *array* masukan merupakan *array* berukuran $M \times N$ yang berisi nilai piksel pada koordinat yang bersesuaian yang kemudian dibentuk vektor sehinggal menjadi satu baris. Untuk citra berwarna, bangun blok piksel dengan ukuran $3b$ piksel yang tediri dari piksel merah, hijau, dan biru, kemudian lakukan proses PCA.

Setelah pembuatan *array* masukan selesai, tentukan nilai N_t yang merupakan jumlah dimensi yang akan diambil. Tahap ini memiliki beberapa langkah yakni menghitung matriks kovarian dari *array* $\vec{x}$ dengan Persamaan 3.2, kemudian menghitung *eigenvector* $\overrightarrow{e_j}$ dan *eigenvalue* λ_j sehingga memenuhi Persamaan 3.3.

$$C = \sum_{i=1}^{N_b} \overrightarrow{x_i}\,\overrightarrow{x_i}^{\,T} \tag{3.2}$$

$$C\overrightarrow{e_j} = \lambda_j \overrightarrow{e_j} \tag{3.3}$$

Menggunakan *eigenvector* $\vec{e}$, dapat ditentukan *principal component* dengan membentuk matriks satu dimensi yang baru untuk setiap blok citra $\overrightarrow{x_i}$ menggunakan persamaan 3.4, dengan $j = 1 \dots, b$ dan $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_b$, dimana $a_j = \overrightarrow{x_i}^T \overrightarrow{e_j}$ dan $\overrightarrow{a_i} = (a_1 \dots a_b)$ merupakan *principal component* yang dapat digunakan sebagai representasi data yang baru.

$$\overrightarrow{x_i} = \sum_{j=1}^{b} a_j \overrightarrow{e_j} \tag{3.4}$$

Kemudian simpan data koordinat blok citra bersama dengan *principal component* milik matriks $\vec{a}$ kedalam matriks $\vec{s}$. Masuknya data koordinat blok citra dan *principal component* menandakan berakhirnya tahap pertama proses pertama.

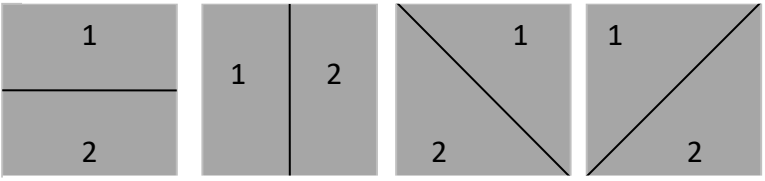
Tahap pertama proses kedua yakni penghitungan aturan perbandingan nilai piksel, seluruh kumpulan blok citra yang berisi sejumlah *b* piksel akan diproses sebagai berikut:

Jika citra masukan memiliki *colorpsace* selain *grayscale*, maka rubah *colorspace* citra tersebut menjadi RGB. Hitung tiga karakteristik pertama (c_1, c_2, c_3), dimana c_1 merupakan fitur rataan jumlah total piksel yang berada pada piksel merah, c_2 merupakan fitur rataan jumlah total piksel yang berada pada piksel hijau, c_3 merupakan fitur rataan jumlah total piksel yang berada pada piksel biru. Jika citra masukan memiliki *colorspace graycale*, maka nilai c_1, c_2, dan c_3 adalah nol.

Rubah *colorspace* blok citra menjadi *Y channel* dengan mengganti nilai setiap piksel dengan nilai baru yang didapat dari Persamaan 3.5, dimana R, G, dan B adalah nilai piksel merah, hijau, dan biru pada sebuah piksel.

$$Y = 0.299\,R + 0.587G + 0.114B \tag{3.5}$$

Bagi blok citra tersebut kedalam empat macam pola seperti pada Gambar 3.9 yang saling membagi blok citra menjadi dua bagian yang sama.



Pola C4 Pola C5 Pola C6 Pola C7

Gambar 3.9 Pola daerah pembagian blok citra

Hitung empat karakteristik selanjutnya (c_4, c_5, c_6, c_7) dengan Persamaan 3.6 sesuai pada masing-masing pola yang diberikan pada Gambar 3.9. Setiap hasil perhitungan tersebut

kemudian disimpan dengan cara disisipkan pada matriks $\vec{s}$ pada elemen yang bersesuaian dengan koordinat blok citra.

$$c_i = sum\big(part1(i)\big)/sum\big(part1(i) + part2(i)\big) \quad (3.6)$$

Setelah tahap pertama yang terdiri dari dua proses tersebut selesai, maka akan didapatkan sebuah kontainer $\vec{s}$ berisi kumpulan pasangan data berupa koordinat blok citra yang direpresentasikan dengan koordinat ujung kiri atas dari blok citra tersebut, serta hasil *principal component* dan fitur perbandingan nilai piksel yang sudah didapat. Contoh data keluaran tersebut dapat dilihat pada Gambar 3.10.

Tahap ketiga, kontainer $\vec{s}$ tersebut kemudian di urutkan menggunakan metode *lexicographical sorting* berdasarkan nilai *principal component* dan fitur perbandingan nilai piksel pada setiap blok citra. Setelah pengurutan, maka blok citra dengan *principal component* dan fitur perbandingan nilai piksel yang sama atau bermiripan akan terletak berdekatan.

```
[(0, 0), [192, 199, 212], [0.99452807710343727,
[(0, 1), [192, 199, 212], [0.99442599058010706,
[(0, 2), [192, 199, 212], [0.99424857203982842,
[(0, 3), [192, 199, 212], [0.99413887619756902,
[(0, 4), [192, 199, 211], [0.99391929193715634,
[(0, 5), [192, 199, 211], [0.99386293336933906,
[(0, 6), [192, 199, 211], [0.99366037117549344,
 [(0, 7), [192, 199, 211], [0.9934739931733193,
[(0, 8), [192, 199, 211], [0.99322226758263354,
[(0, 9), [192, 199, 211], [0.99303073893028226,
```

Gambar 3.10 Contoh potongan hasil pencarian fitur dari 10 *overlapping block* dengan *principal component analysis* dan aturan perbandingan nilai piksel

Tahap keempat, buat sebuah matriks baru $\vec{t}$ dan mengisinya dengan pasangan koordinat blok citra (x_i, y_i) dan (x_j, y_j) hanya jika $|i - j| < N_n$, dimana i dan j merupakan indeks blok citra

yang berada pada kontainer $\vec{s}$ dan N_n merupakan batas maksimal jarak tetangga yang diperhitungkan.

Tahap kelima, hitung *offset* dari setiap elemen matriks $\vec{t}$ tersebut dengan Persamaan 3.7.

$$offset = (x_i - x_j, y_i - y_j) \tag{3.7}$$

Tahap keenam, buang semua pasangan koordinat pada matriks $\vec{t}$ yang memiliki frekuensi *offset* kurang dari N_f.

$$N_d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.8}$$

Tahap ketujuh, buang semua pasangan koordinat pada matriks $\vec{t}$ yang memiliki nilai *offset* kurang dari N_d, dimana nilai *offset* dihitung dengan Persamaan 3.8.

Tahap terakhir, buat citra *ground truth* dengan membuat citra biner (hitam putih) dengan nilai standar 0, dan beri nilai 1 pada setiap lokasi pasangan blok citra yang tersisa. Dari citra *ground truth* tersebut, buat duplikat citra masukan dan beri garis berwarna pada setiap tepi dari daerah putih pada *ground truth* untuk mendapatkan citra hasil deteksi.

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 *duplication detection, robust detection,* dan *robust-duplication detection* 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) i3-3.4 GHz
	Memori	4 GB 798.1 MHz DDR3
Perangkat Lunak	Sistem Operasi	Windows 10 64 bit
	Perangkat Pengembang	PyCharm 4.5 Professional Edition
	Bahasa Pemrograman	Python versi 2.7

4.2 Implementasi

Pada subbab implementasi ini menjelaskan mengenai pembangunan perangkat lunak secara detail dan menampilkan *pseudocode* yang digunakan mulai tahap pembuatan *overlapping block*, penghitungan fitur karakteristik setiap metode, hingga pendeteksian serangan *copy-move* pada citra menggunakan

gabungan metode *duplication detection* dan *robust detection* yang kemudian diberi nama *robust-duplication detection*.

4.2.1 Metode *Robust-Duplication Detection*

Metode *robust-duplication detection* akan memanfaatkan tiga buah objek utama bernama *Blocks*, *Container*, dan *ImageObject* yang dibuat khusus untuk menjalankan deteksi *copy-move*. Secara singkat, objek *Blocks* menampung sebuah blok citra dan mencari fitur karakterstiknya, objek *Container* berlaku sebagai kontainer penampung data hasil perhitungan, dan objek *ImageObject* menangani deteksi sebuah citra masukan yang kemudian melibatkan beberapa objek *Block*s dan *Container*.

Objek *Blocks* merupakan objek yang berfungsi menampung sebuah blok citra, memiliki fungsi untuk menghitung dua jenis fitur karakteristik berupa *principal component*, dan nilai perbandingan piksel dari blok citra yang dimasukkan. *Pseudocode* fungsi *__init__* selaku konstruktor dari objek ini terdapat pada *Pseudocode* 4.1. Pada fungsi konstruktor ini, dilakukan inisialisasi berupa penyimpanan blok citra kedalam variabel *image* pada baris 4. Dilakukan juga perubahan *colorspace* jika blok citra yang masuk memiliki *colorpsace* selain *grayscale* yang terdapat pada pada baris 5 hingga 7. Kemudian pada baris 8 dilakukan *load* piksel kedalam memori agar dapat dilakukan pengambilan nilai piksel secara langsung menggunakan koordinat *x* dan *y*. Pada baris 9, data koordinat blok citra dibentuk menjadi tipe data *tuple* dan disimpan pada variabel *coor*. Pada baris 10 dan 11 dilakukan penyimpanan data panjang dan lebar dari blok citra dan dimensi blok citra atau *blockDimension*.

Objek *Container* meruapakan objek yang berfungsi menampung data perhitungan fitur karakteristik dan data pasangan blok citra. *Pseudocode* fungsi *__init__* yang menjelaskan proses yang dilakukan pada saat objek *Container* dibuat dapat dilihat pada *Pseudocode* 4.2. Pada baris 3, dilakukan inisialisasi variabel

bernama *container* yang diisi dengan tipe data *python list* yang masih kosong.

```
1   CLASS Blocks(object):
2       FUNCTION __init__(self,
                          image,
                          x, y,
                          blockDimension):

            # Fungsi konstruktor untuk persiapan citra masukan
4           image <- image
5           IF  image.mode != 'L':
6                image <- image.convert('RGB')
7           ENDIF

8           pixels <- image.load()
9           coor <- (x, y)
10          imageWidth, imageHeight <- image.size
11          blockDimension <- blockDimension
12      ENDFUNCTION
```

***Pseudocode* 4.1 Fungsi *__init__* untuk inisialisasi variabel pada objek Blocks**

```
1   CLASS Container(object):
2       FUNCTION __init__(self):
            # Fungsi konstruktor untuk menginisialisasi list
3           container <- []
4           RETURN
5       ENDFUNCTION
```

***Pseudocode* 4.2 Fungsi *__init__* untuk menginisialisasi variabel container**

Pada objek *ImageObject*, proses yang pertama kali dilakukan pada saat metode ini dibuat adalah menginisialisasi variabel yang akan digunakan pada proses pendeteksian serangan *copy-move* terhadap citra. Proses inisialisasi ini dilakukan pada fungsi *__init__* yang berlaku sebagai konstruktor objek *ImageObject* dan akan selalu dijalankan saat terjadi pembuatan objek tersebut. Parameter yang disiapkan meliputi parameter dari metode *duplication detection*, parameter dari metode *robust detection*, dan parameter citra masukan. Parameter algoritma berperan sebagai *threshold* dalam pengambilan keputusan, sementara parameter citra berperan menyimpan identitas citra masukan sepreti *path* masukan, nama citra, *path* keluaran, dan

ukuran blok citra yang ingin digunakan. Inisialisasi dilakukan dengan memberikan nilai numerik pada parameter algoritma. *Pseudocode* fungsi *__init__* terdapat pada *Pseudocode* 4.3. Fungsi ini membutuhkan modul *Container* untuk dapat berjalan.

Pertama, pada baris 4 hingga 7 dilakukan penerimaan argumen yang terkait dengan citra masukan yakni berupa tempat citra masukan berada atau *imageDir*, nama berkas citra atau *imageName*, tempat citra keluaran diinginkan atau *targetResult*. Kemudian dilakukan penentuan atribut citra masukan dengan menghitung panjang dan lebar citra masukan dan disimpan pada variabel *imageWidth* dan *imageHeight*.

Tahap kedua, pada baris 8 hingga 14 dilakukan inisialisasi parameter metode pertama yakni *duplication detection* dengan menentukan jumlah piksel pada sebuah citra atau *N*, lebar sisi blok citra yang akan digunakan atau *blockDimension*, jumlah piksel yang terdapat pada blok citra atau *b*, jumlah blok citra yang dapat diekstrak dari citra masukan atau *Nb*, jumlah tetangga terdekat yang dianalisa saat membuat pasangan blok citra atau *Nn*, jumlah minimal frekuensi *offset* dari pasangan blok citra yang diterima sebagai daerah *copy-move* atau *Nf*, dan *threshold* minimal *offset* sebuah pasangan blok citra atau *Nd*.

Tahap ketiga, pada baris 15 hingga 17 dilakukan inisialisasi parameter metode kedua yakni *robust detection* dengan menentukan *threshold* minimal selisih masing-masing fitur karakteristik dengan tipe data *tuple* berisi tujuh nilai numerik yang merujuk pada masing-masing fitur karakteristik kemudian menyimpannya pada variabel *P*, dilanjutkan dengan penentuan *threshold* maksimal jumlah pada selisih fitur c_1, c_2, dan c_3 (*t1*), dan penentuan *threshold* maksimal jumlah pada selisih fitur c_4, c_5, c_6, dan c_7 (*t2*).

Tahap keempat, pada baris 18 hingga 20 dilakukan inisialisasi variabel kontainer data *featureContainer* untuk menyimpan properti blok citra seperti koordinat dan fitur karakteristiknya, kontainer data *pairContainer* untuk menyimpan data pasangan blok citra seperti koordinat masing-masing blok

citra dan *offset* pasangan blok citra. Kedua kontainer diatas dibuat dengan menginstansiasi kelas *Container* yang dibuat khusus untuk menampung jenis data properti dari blok citra dan memiliki fungsi yang mencukupi untuk manajemen data blok citra. Terakhir kontainer data *offsetDict* bertipe *python dictionary* untuk menyimpan frekuensi *offset* dari pasangan blok citra.

```
      import Container

1     CLASS ImageObject(object):
2         FUNCTION __init__(self,
                            imageDir,
                            imageName,
                            blockDimension,
                            targetResult):
              # Fungsi konstruktor untuk mempersiapkan citra dan
              # parameter algoritma

3             OUTPUT imageName

              # parameter gambar
4             targetResult <- targetResult
5             imagePath <- imageName
6             image <- Image.open(imageDir+imageName)
7             imageWidth,  imageHeight <- image.size

              # parameter algoritma duplication detection
8             N <- imageWidth *  imageHeight
9             blockDimension <- blockDimension
10            b <- blockDimension *  blockDimension
11            Nb <- (imageWidth- blockDimension+1)*\
                (imageHeight- blockDimension+1)
12            Nn <- 2
13            Nf <- 750
14            Nd <- 50

              # parameter algoritma robust detection
15            P <- (1.80, 1.80, 1.80, 0.0125, 0.0125,
                 0.0125, 0.0125)
16            t1 <- 2.80
17            t2 <- 0.02

              # inisialisasi kontainer untuk menampung data
18            featureContainer <- Container.Container()
19            pairContainer <- Container.Container()
20            offsetDict <- {}
21        ENDFUNCTION
```

Pseudocode* 4.3 Fungsi *__init__* untuk inisialisasi variabel yang digunakan pada metode *robust-duplication detection

Selesainya proses inisialisasi akan dilanjutkan oleh pelaksanaan fungsi utama, bernama fungsi *run* yang terdapat pada *Pseudocode* 4.4. Fungsi ini memanggil serangkaian fungsi penunjang berjalannya deteksi *copy-move* pada baris 3 hingga 6 seperti *compute* untuk menghitung fitur karakteristik, *sort* untuk mengurutkan kontainer data berdasarkan fitur karakteristik, *analyze* untuk mendapatkan pasangan blok citra yang diduga identik akibat serangan *copy-move* lalu menghapus sisanya yang tidak sesuai dengan *threshold*, dan terakhir fungsi *reconstruct* untuk melakukan rekonstruksi ulang citra dan mengeluarkan citra hasil pendeteksian.

```
1   CLASS ImageObject(object):
2       FUNCTION run(self):
            # Fungsi untuk menjalankan serangkaian proses
            # algoritma

3           compute()
4           sort()
5           analyze()
6           reconstruct()
7       ENDFUNCTION
```

***Pseudocode* 4.4 Fungsi *run* untuk memanggil serangkaian fungsi tahapan algoritma**

Tahap selanjutnya mulai masuk pada proses pertama pada tahap inti metode *robust-duplication detection*. **Proses pertama** adalah *compute* yang mempersiapkan *overlapping block* dan memberikan setiap blok citra kepada objek *Block* untuk kemudian dihitung fitur karakteristiknya. *Pseudocode* fungsi *compute* terdapat pada *Pseudocode* 4.5.

Pada baris 3 hinga 6, dilakukan dua kali perulangan sebanyak panjang *overlapping block* yang ada yang dihitung dengan Persamaan 2.1. Pada baris 5, dilakukan pemotongan citra masukan dengan memanggil fungsi *crop* pada objek *image* dengan batas yang sesuai dengan aturan *overlapping block*. Hasil pemotongan tersebut kemudian disimpan pada variabel *tmpImage*. Pada baris 6, dibuat variabel *imageBlock* yang akan diisi dengan sebuah objek *Blocks* yang dibuat dengan memasukkan koordinat

blok citra dari *tmpImage* dan *tmpImage* itu sendiri yang berwujud citra, serta lebar blok citra yang akan digunakan pada variabel *blockDimension*. Pembuatan objek blok ini merujuk pada fungsi *__init__* milik objek *Block* yang terdapat pada *Pseudocode* 4.1. Pada baris 7, kode memanggil variabel *featureContainer* selaku kontainer data untuk memasukkan sebuah data blok citra dengan fungsi *addBlock* yang terdapat pada *Pseudocode* 4.10. Data blok citra tersebut dihitung dengan memanggil fungsi *computeBlock* pada objek *Blocks*.

```
      import Blocks

1     CLASS ImageObject(object):
2         FUNCTION compute(self):
              # Fungsi untuk pembuatan overlapping block

3             for i in range(0,  imageWidth -  blockDimension + 1):
4                 for j in range(0,
                                 imageHeight -  blockDimension +
                                 1):
5                     tmpImage <- image.crop((i,
                                              j,
                                              i+ blockDimension,
                                              j+ blockDimension))
6                     imageBlock <- Blocks.Blocks(tmpImage,
                                                  i,
                                                  j,
                                                  blockDimension)
7                     featureContainer.addBlock(
8                     imageBlock.computeBlock())
9                 ENDFOR
10            ENDFOR
11        ENDFUNCTION
```

***Pseudocode* 4.5 Fungsi *compute* untuk membuat *overlapping block* dan menyimpan hasil perhitungan fitur karakteristik**

Fungsi *computeBlock* mengatur pembentukan sebuah data blok citra yang berisi koordinat dan fitur karakteristik, kemudian mengembalikan hasil perhitungan dalam bentuk *python list*. Pada baris 3, fungsi ini membuat variabel bertipe *python list* bernama *blockData* untuk diisi dengan serangkaian data seperti koordinat blok citra, dan fitur karakteristik baik dari metode *duplication detection* maupun dari metode *robust detection*. Fitur karakteristik pertama dan kedua dari blok tersebut didapat dengan cara

memanggil fungsi *computeCharaFeature* dan *computePCA*, lalu memasukkan nilai kembalian dari kedua fungsi tersebut kedalam *python list* bernama *blockData* dengan menggunakan fungsi *append* seperti pada baris 6. *Pseudocode* fungsi *computeBlock* terdapat pada *Pseudocode* 4.6.

```
1   CLASS Blocks(object):
2       FUNCTION computeBlock(self):
            # Fungsi untuk membentuk representasi data
            # dari blok citra

3           blockData <- []
4           blockData.append(coor)
5           blockData.append(computeCharaFeatures(4))
6           blockData.append(computePCA(6))
7           RETURN blockData
8       ENDFUNCTION
```

***Pseudocode* 4.6 Fungsi computeBlock untuk membuat data blok citra**

Pada fungsi *computeCharaFeatures* dilakukan penghitungan fitur karakteristik blok citra menggunakan metode *robust detection*. *Pseudocode* fungsi *computeCharaFeature* terdapat pada *Pseudocode* 4.7.

Pada baris 3 hingga 16, dilakukan inisialisasi variabel yang diperlukan untuk menghitung fitur karakteristik. Baris 3 membuat variabel *yImages* untuk menyimpan matriks citra yang telah dirubah *colorspace*nya menjadi *Y channel* (*grayscale*). Baris 4 membuat variabel *yPixels* untuk menyimpan matriks citra sebagai acuan untuk menghitung fitur karakteristik. Baris 5 membuat *python list* sepanjang tujuh elemen yang diisi dengan elemen kosong (*None*) yang kemudian akan digunakan untuk menyimpan tujuh buah fitur karakteristik. Baris 6 hingga 8 membuat variabel untuk menyimpan jumlah nilai piksel pada piksel merah, hijau, dan biru, yakni *sumR*, *sumG*, dan *sumB* yang diberi nilai awal nol. Baris 9 hingga 16 membuat variabel penyimpan jumlah piksel masing-masing pada bagian satu dan dua dari pola yang telah dijelaskan pada Gambar 2.5 yang digunakan untuk menghitung fitur karakteristik keempat hingga ketujuh (c_4, c_5, c_6, c_7).

Pada baris 17 hingga 36, dilakukan penghitungan fitur karakteristik pertama hingga ketiga (c_1, c_2, c_3). Fitur karakteristik pertama dan ketiga merupakan rataan nilai piksel merah, hijau, dan biru pada *colorspace* RGB. Penghitungan tiga karakteristik tersebut dibagi menjadi dua kasus. Kasus pertama pada baris 17 hingga 20, yakni dimana citra masukan memiliki *colorspace grayscale*, karena citra *grayscale* hanya memiliki nilai piksel yang merepresentasikan intensitas warna putih. Nilai nol tersebut kemudian dimasukkan kedalam variabel *python list* bernama *charaFeature* pada indeks pertama, kedua, dan ketiga.

Kasus kedua pada baris 21 hingga 36, yakni dimana citra masukan memiliki *colorspace* RGB. Pada kasus kedua tersebut maka fitur karakteristik pertama hingga ketiga dihitung dengan melakukan dua kali perulangan sebanyak piksel pada blok citra untuk menghitung jumlah total masing-masing piksel merah, hijau, dan biru pada blok citra tersebut, kemudian menyimpannya pada variabel *sumR*, *sumG*, dan *sumB*, seperti yang terdapat pada baris 22 hingga 28. Setelah didapatkan jumlah nilai piksel merah, hijau, dan biru pada sebuah blok citra, proses dilanjutkan dengan membagi jumlah nilai piksel pada tiga warna sebelumnya dengan jumlah piksel pada blok citra tersebut untuk mendapatkan nilai rataan, seperti yang dilakukan pada baris 30 hingga 32. Hasil rataan nilai piksel pada tiga warna tersebut kemudian dimasukkan kedalam variabel *python list* penampung fitur karakteristik bernama *charaFeature*, dengan *sumR*, *sumG*, dan *sumB* menempati indeks pertama, kedua, dan ketiga.

Pada baris 37 hingga 46, dilakukan pengkondisian blok citra agar dapat dihitung keempat fitur karakteristik sisanya. Karena empat fitur karakteristik terakhir dihitung dalam *colorspace grayscale*, maka citra masukan yang memiliki *colorspace* RGB akan dirubah menjadi *colorspace grayscale*. Nilai piksel yang baru didapat dari nilai piksel yang lama yang dihitung dengan Persamaan 2.1. Hasil konversi matriks citra menuju *colorspace grayscale* tersebut kemudian disimpan pada variabel *yPixels* yang akan digunakan pada proses selanjutnya. Pada kasus

jika citra masukan sudah memiliki *colorspace grayscale*, maka tidak dibutuhkan pengubahan *colorspace* menjadi *grayscale*.

Pada baris 47 hingga 105, dilakukan penghitungan empat fitur karakteristik terakhir (*c4*, *c5*, *c6*, *c7*) yang didapatkan dari aturan perbandingan jumlah nilai piksel yang merujuk pada Gambar 2.5, sementara aturan penghitungan keempat fitur tersebut merujuk pada Subbab 2.9. Kembali terdapat dua kasus pada proses ini. Pertama pada baris 47 hingga 75, jika citra masukan memilki *colorspace grayscale*, dan kasus kedua pada baris 76 hingga 105 dimana citra masukan memiliki *corolspace* RGB. Pada kasus pertama, dilakukan dua kali perulangan sebanyak jumlah piksel pada blok citra pada baris 48 hingga 71. Perulangan tersebut dilakukan untuk menghitung jumlah nilai piksel sesuai pada pola tertentu. Perulangan tersebut memiliki banyak percabangan seperti pada baris 50, 55, 59, dan 65 yang mencari termasuk pada anggota apa lokasi piksel yang sedang diiterasi saat ini (*part1* atau *part2*).

Setelah diketahui jenis keanggotaan lokasi pada piksel saat ini, tambah nilai piksel tersebut kepada variabel penyimpan yang sesuai dengan daerah keanggotaan dimana piksel tersebut berada. Pada baris 72 hingga 75, dilakukan pembagian hasil penjumlahan nilai piksel dengan total piksel yang ada pada blok citra tersebut untuk mendapatkan rataan nilai piksel pada pola tertentu. Pada kasus kedua, proses berjalan seperti kasus pertama, hanya saja pengambilan nilai piksel merujuk pada variabel *yPixels* sebagai variabel penampung citra masukan dengan *colorspace grayscale* yang telah dihitung pada tahapan sebelumnya.

Pada baris 106, dilakukan operasi pembulatan untuk memberikan toleransi pada fitur karakteristik dari metode *duplication detection*. Pembulatan dilakukan dengan membulatkan setiap elemen dengan ketelitian yang mengacu pada variabel *roundTo*, memasukkannya kedalam *python list* dan menyimpannya pada variabel *roundedResult*. Variabel roundTo berisi argumen numerik yang diberikan sebagai parameter saat fungsi *computeCharaFeatures* dipanggil.

Kemudian pada baris 107, dilakukan pengembalian data berupa variabel *roundedResult* bertipe py*ton list* tadi kepada pemanggil fungsi.

```
1   CLASS Blocks(object):
2       FUNCTION computeCharaFeatures(self, roundTo):
            # Menghitung 7 nilai fitur karakteristik
            # setiap blok citra berdasar robust detection

            # variable untuk Characteristics Features
3           yImages <- None
4           yPixels <- None
5           charaFeature <- [None] * 7
6           sumR <- 0
7           sumG <- 0
8           sumB <- 0
9           c4 part1 <- 0
10          c4 part2 <- 0
11          c5 part1 <- 0
12          c5_part2 <- 0
13          c6_part1 <- 0
14          c6_part2 <- 0
15          c7_part1 <- 0
16          c7_part2 <- 0

            # Menghitung c1, c2, dan c3 2
17          IF  image.mode = 'L':
18              charaFeature[0] <- 0
19              charaFeature[1] <- 0
20              charaFeature[2] <- 0
21          ELSEIF  image.mode = 'RGB':
22              for y in range(0,  imageHeight):
23                  for x in range(0,  imageWidth):
24                      tmpR, tmpG, tmpB <- pixels[x,y]
25                      sumR += tmpR
26                      sumG += tmpG
27                      sumB += tmpB
28                  ENDFOR
29              ENDFOR
30              sumR <- sumR / (blockDimension* blockDimension)
31              sumG <- sumG / (blockDimension* blockDimension)
32              sumB <- sumB / (blockDimension* blockDimension)
33              charaFeature[0] <- sumR
34              charaFeature[1] <- sumG
35              charaFeature[2] <- sumB
36          ENDIF

            # Merubah colorspace blok menjadi Y Channel
37          IF  image.mode = 'RGB':
38              yImages <- image.convert('L')
39              yPixels <- yImages.load()
40              for y in range(0,  imageHeight):
41                  for x in range(0,  imageWidth):
42                      tmpR, tmpG, tmpB <- pixels[x,y]
```

```
43                      yPixels[x,y] <- int(0.299 * tmpR) + \
                                        int(0.587 * tmpG) + \
                                        int(0.114 * tmpB)
44                  ENDFOR
45              ENDFOR
46          ENDIF

            # Menghitung c4, c5, c6, dan c7
47          IF  image.mode = 'L':
48              for y in range(0,  imageHeight):
49                  for x in range(0,  imageWidth):
                        # c4
50                      IF y <=  imageHeight / 2:
51                          c4 part1 +=  pixels[x,y]
52                      ELSE:
53                          c4 part2 +=  pixels[x,y]
54                      ENDIF
                        # c5
55                      IF x <=  imageHeight / 2:
56                          c5 part1 +=  pixels[x,y]
57                      ELSE:
58                          c5_part2 +=  pixels[x,y]
59                      ENDIF
                        # c6
60                      IF x-y >= 0:
61                          c6_part1 +=  pixels[x,y]
62                      ELSE:
63                          c6 part2 +=  pixels[x,y]
64                      ENDIF
                        # c7
65                      IF x+y <=  blockDimension:
66                          c7 part1 +=  pixels[x,y]
67                      ELSE:
68                          c7 part2 +=  pixels[x,y]
69                      ENDIF
70                  ENDFOR
71              ENDFOR
72              charaFeature[3] <- float(c4_part1) / \
                                   float(c4_part1 +  c4_part2)
73              charaFeature[4] <- float(c5 part1) / \
                                   float(c5 part1 +  c5 part2)
74              charaFeature[5] <- float(c6_part1) / \
                                   float(c6_part1 +  c6_part2)
75              charaFeature[6] <- float(c7_part1) / \
                                   float(c7 part1 +  c7 part2)
76          ELSEIF  image.mode = 'RGB':
77              for y in range(0,  imageHeight):
78                  for x in range(0,  imageWidth):
                        # c4
79                      IF y <=  imageHeight / 2:
80                          c4_part1 +=  yPixels[x,y]
81                      ELSE:
82                          c4 part2 +=  yPixels[x,y]
83                      ENDIF
                        # c5
84                      IF x <=  imageHeight / 2:
85                          c5_part1 +=  yPixels[x,y]
```

```
86                      ELSE:
87                          c5 part2 +=  yPixels[x,y]
88                      ENDIF
                        # c6
89                      IF x-y >= 1:
90                          c6_part1 +=  yPixels[x,y]
91                      ELSE:
92                          c6 part2 +=  yPixels[x,y]
93                      ENDIF
                        # c7
94                      IF x+y <=  blockDimension:
95                          c7_part1 +=  yPixels[x,y]
96                      ELSE:
97                          c7 part2 +=  yPixels[x,y]
98                      ENDIF
99                  ENDFOR
100             ENDFOR
101             charaFeature[3] <- float(c4_part1) / \
                                   float(c4_part1 +  c4_part2)
102             charaFeature[4] <- float(c5 part1) / \
                                   float(c5 part1 +  c5 part2)
103             charaFeature[5] <- float(c6_part1) / \
                                   float(c6_part1 +  c6_part2)
104             charaFeature[6] <- float(c7_part1) / \
                                   float(c7_part1 +  c7_part2)
105         ENDIF

106         roundedResult <- [round(elem, roundTo)
                              for elem in  charaFeature
                              ENDFOR]
107         RETURN roundedResult
108     ENDFUNCTION
```

Pseudocode* 4.7 Fungsi *computeCharaFeatures* untuk menghitung fitur karakteristik berdasarkan *metode robust-detection

Pada fungsi *computePCA*, dilakukan penghitungan fitur karakteristik blok citra menggunakan metode *duplication detection*. *Pseudocode* fungsi *computePCA* terdapat pada *Pseudocode* 4.8. Pada baris 3, dilakukan pembuatan matriks bertipe *numpy array* dari variabel citra masukan *image*. Pada baris 4, dilakukan pemilihan komponen yang disisakan dengan memberikan nilai *integer* pada variabel *n_components*. Variabel *n_components* merepresentasikan jumlah komponen yang ingin diambil pada proses PCA. Pada baris 5 hingga 8, dilakukan penghitungan *principal component* jika citra masukan bertipe *grayscale* dengan tanda memiliki matriks berdimensi dua. Pada

baris 9 hinga 17, dilakukan penghitungan *principal component* jika citra masukan bertipe RGB dengan tanda memiliki matriks berdimensi tiga. Penghitungan *principal component* dilakukan dengan memanggil fungsi *pca.fit_transform* dengan menyertakan matriks masukan kedalamnya untuk menghitung *principal component* dari data masukan, seperti yang terdapat pada baris 6.

Khusus perhitungan pada matriks tiga dimensi, maka dilakukan persiapan data sebelum penghitungan *principal component* dengan cara membentuk ulang matriks data citra baik pada *colorspace* merah, hijau, dan biru agar menjadi matriks dua dimensi dengan panjang tiga kali lebih besar dari panjang citra awal. Pembentukan ulang matriks ini dilakukan dengan bantuan fungsi *concatenate* pada *numpy*, seperti yang terdapat pada baris 10 hingga 13. Setelah matriks sudah menjadi dua dimensi, dilakukan penghitungan *principal component* dengan memanggil fungsi *pca.fit_transform* pada baris 14. Hasil *principal component* tersimpan pada variabel *components_* sehingga isi variabel tersebut akan dibentuk menjadi *python list* dan disimpan pada variabel *principalComponents*. Sebelum mengembalikan hasil, terlebih dulu dilakukan proses pembulatan pada *principal component* dengan fungsi *round* untuk memberikan toleransi seperti yang telah dibahas pada Bab 3. Proses pembulatan dilakukan pada setiap fitur dengan ketelitian sesuai dengan isi variabel *roundTo* yang didapatkan dari argumen saat fungsi dipanggil.

Pseudocode fungsi *pca* untuk menghitung *principal component* terdapat pada *Pseudocode* 4.9. Fungsi ini menerima argumen berupa matriks dua dimensi yang ingin dicari principal componentnya, atau *X*. Pada baris 2, dilakukan pengambilan properti data berupa jumlah data dan dimensi data yang disimpan pada variabel *num_data* dan *dim* melalui pemanggilan fungsi *shape*. Pada baris 3 hingga 6, dilakukan *center data* dengan mencari rataan pada data kemudian data masukan dikurangkan dengan rataan tersebut agar memiliki kondisi *zero mean*. Pada baris 7, dilakukan pencarian matriks kovarian dengan melakukan operasi perkalian *dot* antara data masukan *X* dengan *transpose* dari

data masukan atau *X.T*. Pada baris 8, dilakukan pencarian *eigenvalue* dan *eigenvector* dengan memanfaatkan fungsi *numpy.linalg.eig* dan masing-masing disimpan pada variabel *e*, dan *EV*.

Pada baris 9, dibuat *projection matrix* dengan melakukan operasi perkalian *dot* antara matriks *transpose* dari data masukan atau *X.T* dengan *eigenvalue* yang didapat sebelumnya atau *EV*. Matriks hasil perkalian tersebut dilakukan operasi *transpose* lalu disimpan pada variabel *tmp*. Pada baris 10, matriks *tmp* dibalik urutannya dan disimpan kedalam variabel *V*. Variabel *V* ini menyimpan *projection matrix* atau *principal component* dari data masukan mulai dari yang paling penting. Pada baris 11, dihitung varian dengan melakukan operasi akar pada *eigenvalue e*, yang kemudian sekaligus dibalik urutannya dan disimpan pada variabel *S*. Pembalikan urutan pada variabel *V* dan *S* dilakukan untuk membuat urutan *principal component* dan *varian* yang urut dimulai dari *eigenvalue* yang terbesar menuju ke yang terkecil, dengan kata lain dari yang paling penting menuju ke yang tidak paling penting.

Sebelum berlanjut ke proses kedua, telah didapat data blok citra berupa koordinat dan kedua fiturnya. Data tersebut kemudian disimpan dalam kontainer data *featureContainer* untuk diolah pada proses kedua. Proses pemasukan data tersebut kedalam kontainer data dilakukan oleh fungsi *addBlock*, seperti yang terdapat pada *Pseudocode* 4.10. Pada baris 3 dilakukan penyimpanan data blok citra berupa *blockData* kedalam variabel *container* selaku kontainer data menggunakan fungsi *python list append*.

Setelah selesai fungsi *compute*, metode berlanjut pada **proses kedua** yakni mengurutkan data yang didapat berdasarkan fitur karakteristiknya. Pengurutan ini dilakukan dengan menjalankan fungsi *sort* yang terdapat pada *Pseudocode* 4.11. Pada baris 3, dilakukan pemanggilan fungsi *sortFeatures* milk *featureContainer* dengan tipe objek *Container*. Pada *Pseudocode* 4.12, terdapat fungsi *sortFeatures* yang melakukan pengurutan kontainer data. Fungsi tersebut melakukan pengurutan pada baris 3 dengan memanggil fungsi *sorted* yang menerima masukan berupa

kontainer data yang ingin diurutkan dan aturan pengurutan. Aturan pengurutan tersebut dibuat dengan fungsi anonim *lambda* yang menunjukkan bahwa pengurutan akan dilakukan berdasarkan indeks pertama dilanjutkan indeks kedua secara *lexicographic*.

```
    import numpy as np
    from sklearn.decomposition import PCA

1   CLASS Blocks(object):
2   FUNCTION computePCA(self, roundTo):
            # Fungsi untuk menghitung principal component
            # dari blok citra

3           imageArray <- np.array( image )
4           pca <- PCA(n components=7)
5           IF imageArray.ndim = 2:
6               pca.fit(imageArray)
7               pca.transform(imageArray)
8               RETURN pca.components_
9           ELSE:
10              r <- imageArray[:,:,0]
11              g <- imageArray[:,:,1]
12              b <- imageArray[:,:,2]
13              concat <- np.concatenate(
                    (r,np.concatenate(
                        (g,b),axis=0)),
                    axis=0)
14                  pca.fit transform(concat)
15                  principalComponents = pca.components_
16
                roundedResult <- [round(elem, roundTo)
                                  for elem in pcaResult
17                                ENDFOR]
18              RETURN roundedResult
19          ENDIF
        ENDFUNCTION
```

Pseudocode* 4.8 Fungsi *computePCA* untuk menghitung *principal component* dari sebuah blok citra berdasarkan metode *duplication detection

Setelah selesai fungsi *sort*, metode berlanjut pada **proses ketiga** yakni menganalisa pasangan blok citra dengan penghapusan blok citra yang tidak sesuai dengan aturan metode. Proses analisa tersebut dilakukan dengan memanggil fungsi *analyze* yang berfungsi untuk menganalisa pasangan blok citra. Proses analisa pada fungsi *analyze* terdapat pada *Pseudocode* 4.13.

```
      import numpy as np

1     FUNCTION pca(X):
          # Principal Component Analysis

          # mengambil properti dari data masukan
2         num_data, dim <- X.shape

          # center data
3         mean_X <- X.mean(axis=0)
4         for i in range(num_data):
5             X[i] -= mean_X
6         ENDFOR

          # menghitung covariance matrix
7         M <- np.dot(X, X.T)

          # menghitung eigenvalue dan eigenvector
8         e, EV <- np.linalg.eig(M)

          # membuat projection matrix sementara
9         tmp <- np.dot(X.T, EV).T

          # projection matrix dibalik urutannya berdasarkan
          # eigenvalue terbesar
10        V <- tmp[::-1]

          # menghitung varian, sekaligus membalik urutannya
          # berdasarkan eigenvalue terbesar
11        S <- np.sqrt(e)[::-1]

          # Mengembalikan projection matrix, varian, dan rataan
12        RETURN V, S, mean_X
```

***Pseudocode* 4.9 Fungsi *pca* untuk melakukan *principal component analysis* pada data masukan**

```
1     CLASS Container(object):
2         FUNCTION addBlock(self, blockData):
              # Fungsi untuk memasukkan sebuah blok data
              # pada kontainer data

3             container.append(blockData)
4             RETURN
5         ENDFUNCTION
```

***Pseudocode* 4.10 Fungsi *addBlock* untuk menambahkan sebuah data blok citra ke kontainer data**

Pada baris 3 hingga 12, dilakukan dua kali perulangan sebanyak jumlah kombinasi pasangan blok citra yang bisa dibuat dari kontainer data *featureContainer*. Di dalam perulangan

tersebut, dilakukan proses cek pada setiap pasagan blok citra yang terbuat apakah sesuai dengan batasan yang diberikan metode. Pengecekan ini dilakukan dengan fungsi *isValid* yang terdapat pada baris 5. Fungsi *isValid* akan mengembalikan nilai nol jika pasangan blok citra *i* dan *j* tidak cocok, dan mengembalikan nilai *tuple* (1, *offset*) dari pasangan blok citra *i* dan *j* jika pasangan tersebut sesuai dengan *threshold* yang diberikan metode. Pada baris 6, dilakukan pengecekan apakah hasil fungsi *isValid* adalah benar. Jika benar, maka pada baris 7 pasangan blok citra *i* dan *j* serta *offset* miliknya akan dimasukkan kedalam variabel *python dictionary* bernama *offsetDict* yang menampung frekuensi *offset*. Jika salah, maka pada baris 9 akan dilakukan penghentian satu tingkat perulangan dengan perintah *break* agar perulangan dapat langsung lompat menuju kombinasi berikutnya.

```
1   CLASS ImageObject(object):
2       FUNCTION sort(self):
            # Memanggil fungsi sort pada objek Container

3            featureContainer.sortFeatures()
4       ENDFUNCTION
```

***Pseudocode* 4.11 Fungsi *sort* untuk memanggil fungsi pengurutan yang dimiliki kontainer data yang akan mengurutkan data**

```
1   CLASS Container(object):
2       FUNCTION sortFeatures(self):
            # Fungsi sorting untuk mengurutkan kontainer data

3           container <- sorted(container,
                                key=lambda x:(x[1],
                                              x[2]))
4           RETURN
5       ENDFUNCTION
```

Pseudocode* 4.12 Fungsi *sortFeatures* untuk mengurutkan kontainer data berdasarkan fitur karakteristik secara *lexicographic

Pseudocode fungsi *isValid* yang dipanggil pada fungsi *analyze* terdapat pada *Pseudocode* 4.14. Pada Baris 3, dicek apakah jarak *i* dan *j* masih memenuhi batas ketetanggaan *Nn*. Jika

terpenuhi maka blok *i* dan *j* masih tergolong berdekatan dan jika tidak maka blok *i* dan *j* dikatakan sudah berjauhan. Pada baris 4 dan 5, dilakukan pengambilan fitur dari blok *i* dan *j* dan menyimpannya pada variabel *iFeature* dan *jFeature*. Pada baris 6 hingga 12, dilakukan pengecekan apakah selisih tujuh fitur karakteristik melebihi *threshold* yang diberikan. Jika tidak melebihi, maka pada baris 13 dan 14 dilakukan pengecekan lagi apakah jumlah tiga fitur karakteristik pertama (c_1, c_2, c_3) dan jumlah empat fitur karakteristik terakhir (c_4, c_5, c_6, c_7) melebihi *threshold t1* dan *t2* yang diberikan. Pengecekan selisih fitur ini dilakukan untuk membuang pasangan blok citra yang sama sekali tidak mirip karena memiliki selisih fitur karakteristik yang besar. Jika tidak melebihi, maka pada baris 15 hingga 17 dilakukan pencarian *offset* dengan cara menyimpan koordinat blok *i* dan *j* pada variabel *iCoordinate* dan *jCoordiante*, dilanjutkan dengan menghitung *offset* dengan mengurangkan masing-masing koordinat dengan sumbu yang bersesuaian lalu menyimpan hasilnya pada variabel *offset* yang bertipe *python tuple*.

Offset berguna untuk memberikan informasi seberapa jauh jarak antar pasangan blok *i* dan *j* terpisah pada sumbu x dan y. Selesai menghitung *offset*, pada baris 18 dilakukan penghitungan *magnitude* atau besar dari *offset* yang didapat dengan persamaan *euclidean*. *Magnitude* berguna untuk menentukan jarak antara kedua blok *i* dan *j*, yang dicek kembali pada baris 19 apakah lebih besar dari *threshold* yang diberikan. Pengecekan ini dilakukan untuk membuang pasangan blok citra yang berjarak terlalu dekat, sehingga pasangan blok citra berwarna homogen yang terletak berdekatan seperti tekstur rumput atau langit dapat dihilangkan. Jika *magnitude* dikatakan cukup besar, maka blok *i* dan *j* dikatakan *valid* dan barulah *offset* pasangan blok *i* dan *j* tersebut dikembalikan beserta nilai flag 1 yang menandakan bahwa pasangan blok *i* dan *j* adalah valid. Pada kasus dimana pasangan blok *i* dan *j* tidak dapat melewati satu saja percabangan untuk pengecekan tersebut, maka proses akan langsung melompat

menuju baris 32 yang mengembalikan nilai nol sebagai tanda bahwa psaangan blok *i* dan *j* tidak valid.

Pseudocode fungsi *addDict* yang dipanggil pada fungsi *analyze* terdapat pada *Pseudocode* 4.15. Fungsi ini menerima tiga argumen berupa koordinat blok pertama atau *coor1*, koordinat blok kedua atau *coor2*, dan *offset* dari kedua pasangan blok citra tersebut atau *offset*. Pada baris 3, dilakukan pengecekan apakah variabel *dictionary offsetDict* memiliki *key* seperti pada variabel *offset*. Jika sudah ada, maka fungsi cukup menambahkan *coor1* dan *coor2* kedalamnya. Namun jika belum ada, fungsi akan menambahkan *offset* sebagai *key* baru lalu menambahkan *coor1* dan *coor2* kedalam kamus tersebut. Penggunaan *dictionary* disini berfungsi untuk mendapatkan frekuensi *offset* dengan cepat tanpa harus iterasi seluruh data pasangan blok citra untuk menghitung masing-masing frekuensi *offset*.

```
1   CLASS ImageObject(object):
2       FUNCTION analyze(self):
            # Fungsi untuk melakukan analisa pasangan blok citra

3           for i in range(0,  featureContainer.getLength()):
4               for j in range(i+1,
                               featureContainer.getLength()):
5                   result <- isValid(i, j)
6                   IF result[0]:
7                       addDict(featureContainer.container[i][0],
                                featureContainer.container[j][0],
                                result[1])
8                   ELSE:
9                       break
10                  ENDIF
11              ENDFOR
12          ENDFOR
13      ENDFUNCTION
```

***Pseudocode* 4.13 Fungsi *analyze* untuk melakukan analisa terhadap data blok citra**

Setelah selesai fungsi *analyze*, metode berlanjut pada **proses keempat** yakni rekonstruksi citra untuk membuat citra keluaran. Proses ini dilakukan dengan memanggil fungsi *reconstruct*. Fungsi ini berfungsi untuk membuat ulang citra masukan dan menandai daerah yang terduga terserang *copy-move*.

Fungsi ini juga membuat citra berwarna hitam dengan panjang dan lebar sesuai dengan citra masukan, serta menandai daerah yang terduga terserang *copy-move* dengan warna putih, yang digunakan sebagai citra hasil deteksi. Di sisi lain, fungsi ini juga mengeluarkan citra masukan dengan daerah terduga *copy-move* sudah diberi garis tepi. Proses rekonstruksi gambar pada fungsi *reconstruct* terdapat pada *Pseudocode* 4.16.

```
    Import numpy as np

1   CLASS ImageObject(object):
2       FUNCTION isValid(self, i, j):
            # Fungsi untuk mengecek validitas pasangan blok,

3           IF abs(i-j) <  Nn:
4               iFeature <- featureContainer.container[i][1]
5               jFeature <- featureContainer.container[j][1]
6               IF abs(iFeature[0] - jFeature[0]) <  P[0]:
7               IF abs(iFeature[1] - jFeature[1]) <  P[1]:
8               IF abs(iFeature[2] - jFeature[2]) <  P[2]:
9               IF abs(iFeature[3] - jFeature[3]) <  P[3]:
10              IF abs(iFeature[4] - jFeature[4]) <  P[4]:
11              IF abs(iFeature[5] - jFeature[5]) <  P[5]:
12              IF abs(iFeature[6] - jFeature[6]) <  P[6]:
13              IF abs(iFeature[0] - jFeature[0]) +
                 abs(iFeature[1] - jFeature[1]) +
                 abs(iFeature[2] - jFeature[2]) <  t1:
14                  IF abs(iFeature[3] - jFeature[3]) +
                        abs(iFeature[4] - jFeature[4]) +
                        abs(iFeature[5] - jFeature[5]) +
                        abs(iFeature[6] - jFeature[6]) <  t2:

                        # mencari offset
15                      iCoordinate <- \
                            featureContainer.container[i][0]
16                      jCoordinate <- \
                            featureContainer.container[j][0]

17                      offset <- \
                            (iCoordinate[0] -
                             jCoordinate[0],
                             iCoordinate[1] -
                             jCoordinate[1])

                        # menghitung magnitude
18                      magnitude <- \
                            np.sqrt(pow(offset[0], 2) +
                                    pow(offset[1], 2))
19                      IF magnitude >=  Nd:
20                          RETURN 1, offset
21                      ENDIF
22                  ENDIF
```

```
23                ENDIF
24                ENDIF
25                ENDIF
26                ENDIF
27                ENDIF
28                ENDIF
29                ENDIF
30                ENDIF
31            ENDIF
32            RETURN 0,
33        ENDFUNCTION
```

***Pseudocode* 4.14 Fungsi *isValid* untuk mengetahui apakah sebuah pasangan blok citra dikatakan valid oleh aturan yang digunakan**

```
1   CLASS ImageObject(object):
2       FUNCTION addDict(self, coor1, coor2, offset):
            # Fungsi untuk memasukkan data  pasangan
            # blok citra berupa koordinat dan offset

3           IF  offsetDict.has_key(offset):
4                offsetDict[offset].append(coor1)
5                offsetDict[offset].append(coor2)
6           ELSE:
7                offsetDict[offset] <- [coor1, coor2]
8           ENDIF
9       ENDFUNCTION
```

***Pseudocode* 4.15 Fungsi *addDict* untuk menambahkan pasangan data yang valid kedalam kontainer data bertipe kamus (*Dictionary*)**

Pada baris 4, dibuat matriks citra dengan isi awal nol menggunakan fungsi *numpy.zeros* yang memiliki panjang dan lebar sesuai dengan citra masukan yang mengacu pada variabel *imageHeight* dan *imageWidth*. Matriks nol sebagai representasi citra biner tersebut kemudian disimpan pada variabel *imageArray*. Pada baris 5, dilakukan duplikasi matriks citra masukan kedalam variabel *linedImageImage* yang nantinya akan diberi garis tepi pada daerah yang diduga diserang *copy-move*. Pada baris 6 hingga 18, dilakukan perulangan yang melakukan iterasi terhadap variabel *dictionary offsetDict* yang telah dilakukan pengurutan menggunakan fungsi bawaan *sorted* berdasarkan frekuensi *offset* yang terpanjang. Mengingat *offset* diberlakukan sebagai *key* pada *dictionary* dan koordinat blok citra yang memiliki *offset*

bersesuaian dimasukkan sebagai elemen pada *key* tersebut, maka frekuensi *offset* dengan mudah diapatkan berdasar jumlah pasangan koordinat yang terdapat pada sebuah *offset* tersebut.

Pada baris 11 hingga 18, mulai dari setiap koordinat yang berada pada *offset* yang memiliki frekuensi terpanjang dilakukan pemberian warna putih dengan memberi nilai 255 pada piksel pada koordinat yang bersesuaian. Tentu saja, pemberian warna putih dilakukan selebar blok citra yang nilainya dapat diketahui dari variabel *blockDimension* dengan koordinat blok citra sebagai acuan pada bagian kiri atas blok citra. Selesainya perulangan ini akan menghasilkan matriks citra hasil deteksi *imageArray* berbentuk citra biner (hitam putih) yang siap dikeluarkan menjadi berkas citra.

Pada baris 19 hingga 49, dilakukan pemberian garis tepi pada daerah yang diduga diserang *copy-move* dengan acuan variabel *imageArray* yang telah didapatkan. Pemberian garis tersebut dilakukan dengan perulangan sepanjang matriks citra *imageArray* dan mengecek apakah piksel sekarang termasuk piksel tepi. Sebuah piksel dikatakan piksel tepi apabila piksel tersebut memiliki minimal tiga tetangga yang memiliki nilai piksel berbeda dengannya. Pada kasus ini adala apabila piksel sekarang merupakan piksel putih (bernilai 255) dan disekitarnya terdapat paling tidak satu piksel hitam (bernilai 0). Koordinat piksel tepi tersebut digunakan untuk menentukan lokasi penggambaran garis pada matriks citra *linedImage*. Pemberian garis dilakukan dengan memberi warna hijau dengan cara memberi nilai (0, 255, 0) pada piksel RGB yang sesuai pada koordinat piksel tepi yang telah didapatkan sebelumnya. Pemberian warna hijau ini dilakukan untuk semua kasus sudut, seperti jika piksel tepi merupakan tepi atas, tepi bawah, tepi kana, tepi kiri, tepi atas kanan, tepi atas kiri, tepi bawah kanan, dan tepi bawah kiri. Maka didapatkan matriks citra *linedImage* yang siap dikeluarkan menjadi berkas citra.

Pada baris 50 hingga 52, dilakukan penyimpanan citra hasil menjadi berkas citra. Pada baris 50 dilakukan pengambilan waktu sekarang untuk menandakan kapan citra hasil dibuat yang

nantinya digunakan untuk memberi nama citra. Pada baris 51, dilakukan penyimpanan citra hasil yang bertipe hitam putih yang mengacu pada *imageArray*. Pada bais 52, dilakukan penyimpanan citra hasil berdasarkan citra masukan yang telah diberi garis tepi pada daerah terduga serangan *copy-move* yang mengacu pada variabel *linedImage*.

```
      Import scypy.misc
      Import numpy as np

1     CLASS ImageObject(object):
2         FUNCTION reconstruct(self):
              # Fungsi untuk melakukan konstruksi ulang citra
              # dengan menyertakan hasil dugaan deteksi

4             imageArray <- np.zeros((imageHeight,  imageWidth))
5             linedImageImage <- np.array(image)

6             for key in sorted(offsetDict,
                                key=lambda key:   builtin  .len(
                                    offsetDict[key]),
                                    reverse=True):
7                 IF  offsetDict[key].__len__() <  Nf*2:
8                     break
9                 ENDIF
10                OUTPUT key,  offsetDict[key].__len__()

11                for i in range(offsetDict[key].__len__()):
12                    for j in range(offsetDict[key][i][1],
                                     offsetDict[key][i][1]+
                                         blockDimension):
13                        for k in range(offsetDict[key][i][0],
                                         offsetDict[key][i][0]+
                                             blockDimension):
14                            imageArray[j][k] <- 255
15                        ENDFOR
16                    ENDFOR
17                ENDFOR
18            ENDFOR

              # memberi garis pada daerah terduga pemalsuan
19            for x in range(2,  imageHeight-2):
20                for y in range(2,  imageWidth-2):
21                    IF imageArray[x,y] = 255
                        AND (imageArray[x+1,y] = 0
                          OR imageArray[x-1,y] = 0
                          OR imageArray[x,y+1] = 0
                          OR imageArray[x,y-1] = 0
                          OR imageArray[x-1,y+1] = 0
                          OR imageArray[x+1,y+1] = 0
                          OR imageArray[x-1,y-1] = 0
                          OR imageArray[x+1,y-1] = 0):
```

```
22                      IF imageArray[x-1,y] = 0
                          AND imageArray[x,y-1] = 0
                          AND imageArray[x-1,y-1] = 0:
23                          linedImage[x-2:x,y,1] <- 255
24                          linedImage[x,y-2:y,1] <- 255
25                          linedImage[x-2:x,y-2:y,1] <- 255
26                      ELSEIF imageArray[x+1,y] = 0
                          AND imageArray[x,y-1] = 0
                          AND imageArray[x+1,y-1] = 0:
27                          linedImage[x+1:x+3,y,1] <- 255
28                          linedImage[x,y-2:y,1] <- 255
29                          linedImage[x+1:x+3,y-2:y,1] <- 255
30                      ELSEIF imageArray[x-1,y] = 0
                          AND imageArray[x,y+1] = 0
                          AND imageArray[x-1,y+1] = 0:
31                          linedImage[x-2:x,y,1] <- 255
32                          linedImage[x,y+1:y+3,1] <- 255
33                          linedImage[x-2:x,y+1:y+3,1] <- 255
34                      ELSEIF imageArray[x+1,y] = 0
                          AND imageArray[x,y+1] = 0
                          AND imageArray[x+1,y+1] = 0:
35                          linedImage[x+1:x+3,y,1] <- 255
36                          linedImage[x,y+1:y+3,1] <- 255
37                          linedImage[x+1:x+3,y+1:y+3,1] <- 255

38                      ELSEIF imageArray[x,y+1] = 0:
39                          linedImage[x,y+1:y+3,1] <- 255
40                      ELSEIF imageArray[x,y-1] = 0:
41                          linedImage[x,y-2:y,1] <- 255
42                      ELSEIF imageArray[x-1,y] = 0:
43                          linedImage[x-2:x,y,1] <- 255
44                      ELSEIF imageArray[x+1,y] = 0:
45                          linedImage[x+1:x+3,y,1] <- 255
46                      ENDIF
47                  ENDIF
48              ENDFOR
49          ENDFOR
50          timestr <- time.strftime("%Y%m%d_%H%M%S_")
51          scipy.misc.imsave(targetResult+
                              timestr+
                              imagePath,
                              imageArray)
52          scipy.misc.imsave(targetResult+
                              timestr+
                              " linedImage "+
                              imagePath,
                              linedImage)
53      ENDFUNCTION
```

***Pseudocode* 4.16 Fungsi reconstruct untuk membangun ulang citra jawaban dari citra masukan**

Berakhirnya fungsi *reconstruct* menandakan berakhirnya metode *robust-duplication detection* dan telah dikeluarkannya citra sebagai hasil deteksi serangan *copy-move*.

4.2.2 Penghitungan Nilai MSE

Penghitungan nilai MSE terdapat pada fungsi *calculateMSE*. Fungsi ini digunakan untuk mengukur nilai MSE dari dua buah citra. Nilai MSE dari metode ini didapatkan dari pembandingan citra keluaran metode dan citra *ground truth* pada *dataset*. Nilai ini kemudian digunakan untuk mengukur akurasi metode yang digunakan. *Pseudocode* fungsi *calculateMSE* terdapat pada *Pseudocode* 4.17. Fungsi ini menerima argumen berupa matriks citra keluaran dari metode atau *imageA*, dan citra *ground truth* dari *dataset* atau *imageB*. Kedua matriks citra tersebut kemudian dirubah menjadi tipe data *float* lalu dijumlahkan hasil selisih nilai piksel pada koordinat yang bersesuaian, yang terjadi pada baris 2. Setelah didapat jumlah selisih dari perbedaan nilai piksel, dilakukan pembagian dengan jumlah piksel yang ada pada citra tersebut untuk mendapatkan rataan, yang terjadi pada baris 3. Maka didapatkan rataan kuadrat selisih yang kemudian dikembalikan kepada pemanggil fungsi.

```
1  FUNCTION calculateMse(imageA, imageB):
2      meanSquaredError <- np.sum((imageA.astype("float")
                                   - imageB.astype("float"))
                                  ** 2)
3      meanSquaredError /= float(imageA.shape[0]
                                 * imageA.shape[1])
4      RETURN meanSquaredError
5  ENDFUNCTION
```

***Pseudocode* 4.17 Fungsi *calculateMSE* untuk menghitung MSE**

4.2.3 Penghitungan Nilai *Similarity*

Penghitungan *similarity* terdapat pada fungsi *calculateSimilarity*. Fungsi ini digunakan untuk mengukur nilai

similarity dari dua buah citra. Nilai *similarity* dari metode ini didapatkan dari pembandingan citra keluaran metode dan citra *ground truth* pada *dataset*. Nilai ini kemudian digunakan untuk mengukur seberapa mirip citra keluaran metode dengan citra *ground truth* pada *dataset*. *Pseudocode* fungsi *calculateSimilarity* terdapat pada *Pseudocode* 4.18. Seperti fungsi *calculateMSE* pada subbab sebelumnya, fungsi ini juga menerima argumen berupa matriks citra keluaran dari metode atau *imageA*, dan citra *ground truth* dari *dataset* atau *imageB*. Pada baris 2, lebar citra disimpan pada variabel *x* dan *y*. Pada baris 3, dilakukan inisialisasi variabel penyimpan jumlah piksel yang identik dan yang tidak identik yakni *truePixels* dan *falsePixels* dengan nilai nol. Pada baris 4 hingga 12, dilakukan dua kali perulangan sebanyak jumlah piksel pada citra untuk membandingkan nilai piksel dari *imageA* dengan nilai piksel dari *imageB* yang bersesuaian. Piksel bersesuaian yang memiliki nilai sama dikatakan identik (*similar*). Jumlah piksel bersesuaian yang identik disimpan pada variabel *truePixels*, dan yang tidak identik disimpan pada *falsePixels*.

```
1   FUNCTION calculateSimilarity(imageA, imageB):
2       x, y <- imageA.shape[0:2]
3       truePixels, falsePixels <- 0, 0
4       for i in range(x):
5           for j in range(y):
6               IF all(imageA[i,j] != imageB[i,j]):
7                   falsePixels += 1
8               ELSE:
9                   truePixels += 1
10              ENDIF
11          ENDFOR
12      ENDFOR
13      RETURN truePixels/ float(truePixels+falsePixels) \
            * 100
14  ENDFUNCTION
```

Pseudocode* 4.18 Fungsi calculate*Similarity* untuk menghitung *similarity

Implementasi metode yang telah dibuat menggunakan bahasa *python* dapat diakses langsung melalui alamat berikut: https://github.com/rahmatnazali/Image-Copy-Move-Detection.

(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 berbagai tipe serangan *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 hasil pengujian.

5.1 Lingkungan Pengujian

Lingkungan pengujian pada uji coba permasalahan pendeteksian serangan *copy-move* pada citra dengan gabungan metode *duplication detection* dan *robust detection* menggunakan spesifikasi perangkat keras dan perangkat lunak seperti yang terdapat pada Tabel 5.1.

Tabel 5.1 Spesifikasi lingkungan pengujian

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel(R) Core(TM) i3-3.4 GHz
	Memori	4 GB 798.1 MHz DDR3
Perangkat Lunak	Sistem Operasi	Windows 10 64 bit
	Perangkat Pengembang	PyCharm 4.5 Professional Edition
	Bahasa Pemrograman	Python versi 2.7

5.2 Data Pengujian

Subbab ini menjelaskan mengenai data yang digunakan pada uji coba. Data terdiri dari 30 buah citra yang dibuat dari 10 buah citra otentik yang didapat dari Universitas Friedich-Alexander yang memiliki lisensi MIT [21]. Lisensi ini menyatakan bahwa dataset dapat digunakan, di-*copy*, dimodifikasi, disatukan dengan data lain, dipublikasi, dan disebarkan dengan syarat menyertakan sumber dataset yang digunakan.

Dari citra otentik tersebut dilakukan serangan *copy-move* dengan menggunakan perangkat lunak *Photoshop*, dengan detail sebagai berikut:

1. 10 buah citra pertama merupakan citra otentik yang tidak dilakukan serangan sama sekali.
2. 10 buah citra yang dilakukan serangan *copy-move* biasa.
3. 10 buah citra yang dilakukan serangan *copy-move* dan dilakukan proses *post region duplication process*, dimana diambil salah satu proses yakni proses *blurring* yang dilakukan pada tepi daerah yang diduplikasi.

Masing-masing citra berukuran 512×512 piksel tersebut memiliki pasangan citra yang berlaku sebagai *ground truth* dari serangan *copy-move*. Pada *dataset* berlaku asumsi bahwa duplikasi yang terjadi tidak boleh saling tertindih (*overlap*) serta untuk setiap citra hanya terdapat satu daerah yang dilakukan serangan duplikasi.

Selain *dataset* diatas, terdapat juga dua buah citra dari sumber yang sama dengan kondisi sudah dilakukan serangan *copy-move*. Kedua data citra ini mempresentasikan *dataset* publik.

Nama file untuk citra otentik, citra yang diserang *copy-move*, citra yang diserang *copy-move* dengan *blurring*, dan citra yang menjadi *ground truth* dijelaskan dalam Tabel 5.2, dan Tabel 5.3. Sementara nama citra dan citra *ground truth* dari *dataset* publik terdapat pada Tabel 5.4. Data tersebut kemudian diolah dengan algoritma *duplication detection* dan *robust detection* untuk kemudian dilakukan deteksi serangan *copy-move*.

Selanjutnya akan dijelaskan masing-masing keadaan data masukan yang digunakan. Data masukan berisi citra otentik, citra yang diserang *copy-move*, dan citra yang dilakukan *copy-move* disertai proses *blurring*.

Tabel 5.2 Nama citra otentik dan citra *ground truth*

Nama citra otentik	**Nama citra *ground truth***
01_barrier.png	01_result.png
02_cattle.png	02_result.png
03_clean_walls.png	03_result.png
04_fountain.png	04_result.png
05_four_babies.png	05_result.png
06_horses.png	06_result.png
07_knight_moves.png	07_result.png
08_lone_cat.png	08_result.png
09_malawi_png	09_result.png
10_mykene.png	10_result.png

Tabel 5.3 Nama citra yang diserang *copy-move* biasa dan yang diserang *copy-move* disertai proses *blurring*

Nama citra yang diserang *copy-move*	**Nama citra yang diserang *copy-move* dan blurring**
01_barrier_copy.png	01_barrier_blur.png
02_cattle_copy.png	02_cattle_blur.png
03_clean_walls_copy.png	03_clean_walls_blur.png
04_fountain_copy.png	04_fountain_blur.png
05_four_babies_copy.png	05_four_babies_blur.png
06_horses_copy.png	06_horses_blur.png
07_knight_moves_copy.png	07_knight_moves_blur.png
08_lone_cat_copy.png	08_lone_cat_blur.png
09_malawi_copy_png	09_malawi_blur.png
10_mykene_copy.png	10_mykene_blur.png

Tabel 5.4 Nama citra yang diserang *copy-move* biasa dari *dataset* publik

Nama citra	Nama citra *ground truth*
01_giraffe.png	01_giraffe_answer.png
04_cattle.png	04_cattle_answer.png

5.2.1 Citra otentik

Subbab ini menjelaskan tentang citra otentik. Citra ini sama sekali tidak dilakukan serangan dan langsung menjadi data masukan sejak pertama kali diperoleh. Tujuan dari diikutkannya citra jenis ini adalah untuk mengukur *false positive* dari algoritma yang dibuat.

Gambar 5.1 Citra masukan 1 bernama 01_barrier.png

Citra masukan 1, seperti yang terdapat pada Gambar 5.1, mencoba untuk mengukur seberapa baik metode dalam menangani citra langit dan air yang memiliki kecenderungan warna yang homogen yakni biru dan biru tua. Nilai piksel yang berdekatan

tersebut akan mengakibatkan blok citra memiliki komposisi warna yang sama, dan akan akan menjadi tolak ukur apakah masing-masing metode dapat memberi kesimpulan bahwa citra tersebut otentik.

Citra masukan 2, seperti yang terdapat pada Gambar 5.2, mencoba untuk mengukur seberapa baik metode dalam menangani citra rumput yang memiliki pola acak dan berwarna dominan hijau dan cokelat. Citra masukan 3, seperti yang terdapat pada Gambar 5.3, mencoba untuk mengukur seberapa baik metode dalam menangani citra tembok berpola yang memiliki warna tetap yang tersebar dengan pola acak dan dinding homogen yang memiliki warna keabuan.

Citra masukan 4, seperti yang terdapat pada Gambar 5.4, mencoba untuk mengukur seberapa baik metode dalam menangani citra langit berawan berwarna homogen biru putih yang identik dengan warna biru putih pada kolam air dan gedung, serta warna cokelat dari baju yang identik dengan warna rerumputan.

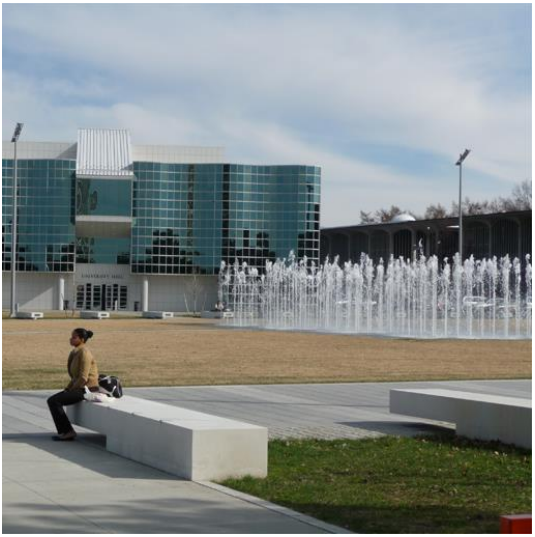
Citra masukan 5, seperti yang terdapat pada Gambar 5.5, mencoba untuk mengukur seberapa baik metode dalam menangani warna kehijauan yang mendominasi citra dan warna kehitaman. Warna homogen yang dominan akan mengakibatkan banyak blok citra yang memiliki komposisi warna sehingga dapat terjadi *false positive*.

Citra masukan 6, seperti yang terdapat pada Gambar 5.6, mencoba untuk mengukur seberapa baik metode dalam menangani warna homogen biru cerah pada citra langit yang sekilas hampir tidak memiliki perbedaan. Citra ini juga mengukur seberapa baik metode dalam menangani pola tekstur pada tembok yang memiliki warna dominan yang sama yakni cokelat putih, namum memiliki bentuk yang berbeda.

Citra masukan 7, seperti yang terdapat pada Gambar 5.7, mencoba untuk mengukur seberapa baik metode dalam menangani citra homogen kekuningan yang dikelilingi oleh citra homogen kuning dan biru tua.

Gambar 5.2 Citra masukan 2 bernama 02_cattle.png

Gambar 5.3 Citra masukan 3 bernama 03_clean_walls.png



Gambar 5.4 Citra masukan 4 bernama 04_fountain.png

Gambar 5.5 Citra masukan 5 bernama 05_four_babies.png

Gambar 5.6 Citra masukan 6 bernama 06_horses.png

Gambar 5.7 Citra masukan 7 bernama 07_knight_moves.png

Gambar 5.8 Citra masukan 8 bernama 08_lone_cat.png

Gambar 5.9 Citra masukan 9 bernama 09_malawi.png

Gambar 5.10 Citra masukan 10 bernama 10_mykene.png

Citra masukan 8, seperti yang terdapat pada Gambar 5.8, mencoba untuk mengukur seberapa baik metode dalam menangani citra keabuan dan kehitaman pada corak jalan yang kasar yang mirip dengan citra bulu kucing.

Citra masukan 9, seperti yang terdapat pada Gambar 5.9, mencoba untuk mengukur seberapa baik metode dalam menangani citra homogen biru muda dari langit dan biru tua dari air laut.

Citra masukan 10, seperti yang terdapat pada Gambar 5.10, mencoba untuk mengukur seberapa baik metode dalam menangani citra bebatuan berpola dengan warna dominan cokelat dan putih.

Dari sepuluh buah citra masukan tersebut, tanpa adanya fitur karakteristik dan pemilihan *threshold* yang baik maka blok citra yang memiliki komposisi warna identik akan cenderung diduga sebagai daerah yang dipalsu, padahal semua citra masukan adalah otentik.

5.2.2 Citra yang diserang *copy-move*

Subbab ini menjelaskan tentang citra yang dilakukan serangan *copy-move* saja. Dari citra otentik, dilakukan pengambilan daerah yang homogen atau yang memiliki warna yang cenderung sama dengan menggunakan aplikasi *select* dari *Photoshop*. Daerah yang dipilih untuk diduplikasi tidak harus berbentuk persegi, melainkan dapat berbentuk sembarang. Daerah citra yang telah dipilih tersebut kemudian dilakukan proses duplikasi dengan fitur *copy* dari *Photoshop*. Hasil dari duplikasi tersebut kemudian digeser dan ditempatkan sedemikian rupa untuk menutupi sebuah objek yang ada pada citra otentik.

Citra masukan 1, seperti yang terdapat pada Gambar 5.11, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah air laut untuk menutupi tiga ekor burung pada daerah air laut yang lain. Citra masukan 1 memiliki *ground truth* seperti yang terlihat pada Gambar 5.12.

Citra masukan 2, seperti yang terdapat pada Gambar 5.13, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah rerumputan untuk menutupi seekor burung pada daerah rumput yang lain. Citra masukan 2 memiliki *ground truth* seperti yang terlihat pada Gambar 5.14.

Citra masukan 3, seperti yang terdapat pada Gambar 5.15, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah tembok berwarna abu-abu untuk menutupi sebagian coretan cat berwarna merah pada daerah tembok abu-abu lain. Citra masukan 3 memiliki *ground truth* seperti yang terlihat pada Gambar 5.16.

Citra masukan 4, seperti yang terdapat pada Gambar 5.17, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah langit berawan untuk menutupi bagian atap dari gedung. Citra masukan 4 memiliki *ground truth* seperti yang terlihat pada Gambar 5.18.

Citra masukan 5, seperti yang terdapat pada Gambar 5.19, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah handuk berwarna hijau yang digunakan sebagai alas kucing untuk menutupi seekor kucing. Citra masukan 5 memiliki *ground truth* seperti yang terlihat pada Gambar 5.20.

Citra masukan 6, seperti yang terdapat pada Gambar 5.21, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah langit untuk menutupi patung burung yang berada ditengah citra. Citra masukan 6 memiliki *ground truth* seperti yang terlihat pada Gambar 5.22.

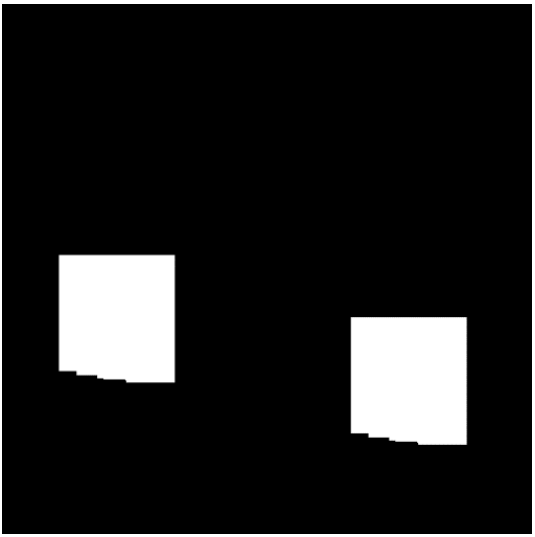
Citra masukan 7, seperti yang terdapat pada Gambar 5.23, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah langit dan awan untuk menutupi sebuah gedung besar yang berada di tengah citra. Citra masukan 7 memiliki *ground truth* seperti yang terlihat pada Gambar 5.24.

Citra masukan 8, seperti yang terdapat pada Gambar 5.25, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah tanah yang berwarna abu-abu dan coklat untuk menutupi tiga seekor kucing yang terletak di sekitar tanah yang berwarna identik. Citra masukan 8 memiliki *ground truth* seperti yang terlihat pada Gambar 5.26.

Citra masukan 9, seperti yang terdapat pada Gambar 5.27, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah langit untuk menutupi seorang yang sedang menaiki perahu. Citra masukan 9 memiliki *ground truth* seperti yang terlihat pada Gambar 5.28.

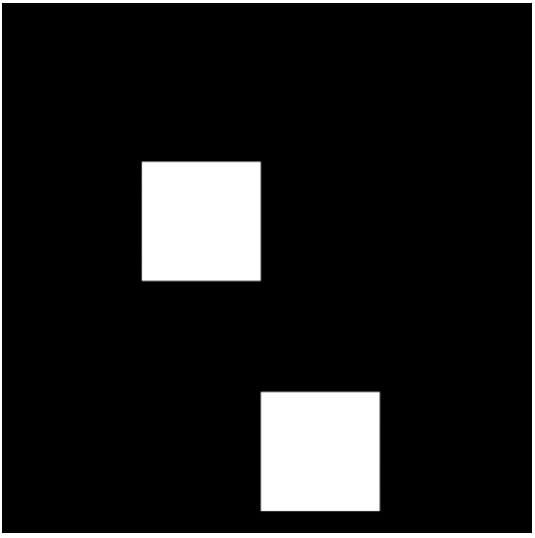
Citra masukan 10, seperti yang terdapat pada Gambar 5.29, mencoba untuk mengukur akurasi metode dalam menangani kasus *copy-move* yang dilakukan dari sebagian daerah dinding batu untuk menutupi pintu masuk dari bangunan. Citra masukan 10 memiliki *ground truth* seperti yang terlihat pada Gambar 5.30.

Gambar 5.11 Citra masukan 1 bernama 01_barrier_copy.png



Gambar 5.12 *Ground truth* citra masukan 1 bernama 01_result.png

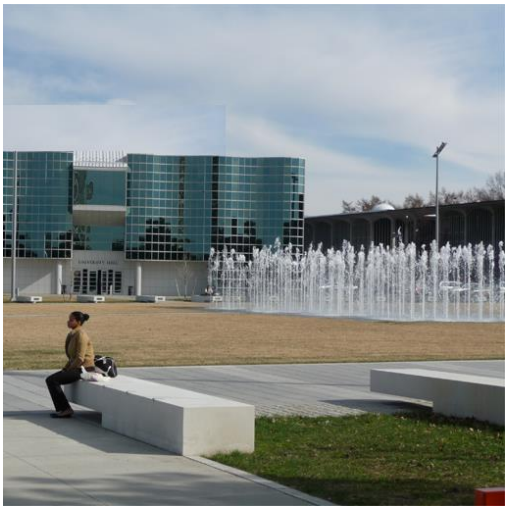
Gambar 5.13 Citra masukan 2 bernama 02_cattle_copy.png



Gambar 5.14 *Ground truth* citra masukan 2 bernama 02_result.png

Gambar 5.15 Citra masukan 3 bernama 03_clean_walls_copy.png

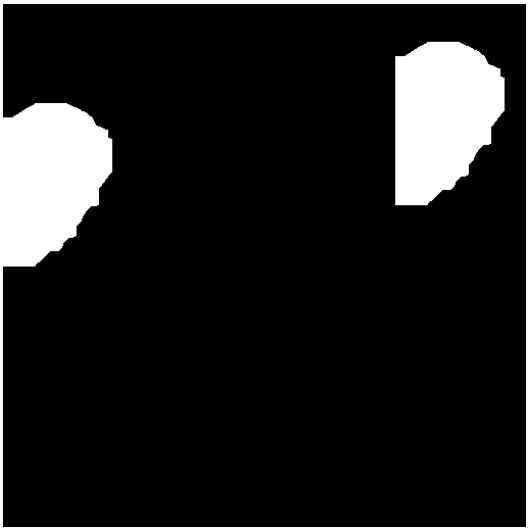
Gambar 5.16 *Ground truth* citra masukan 3 bernama 03_result.png



Gambar 5.17 Citra masukan 4 bernama 04_fountain_copy.png

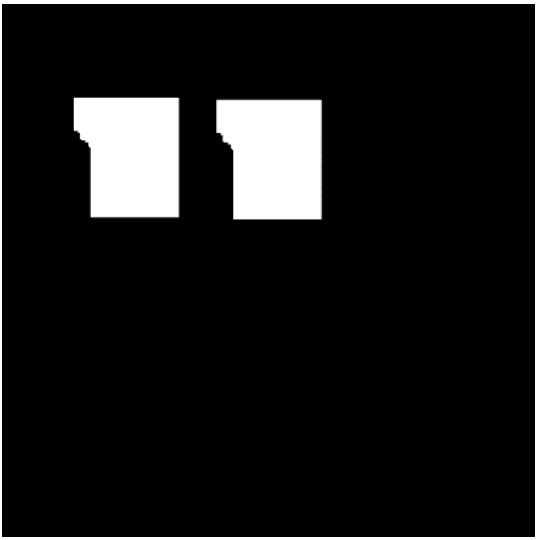
Gambar 5.18 *Ground truth* citra masukan 4 bernama 04_result.png

Gambar 5.19 Citra masukan 5 bernama 05_four_babies_copy.png

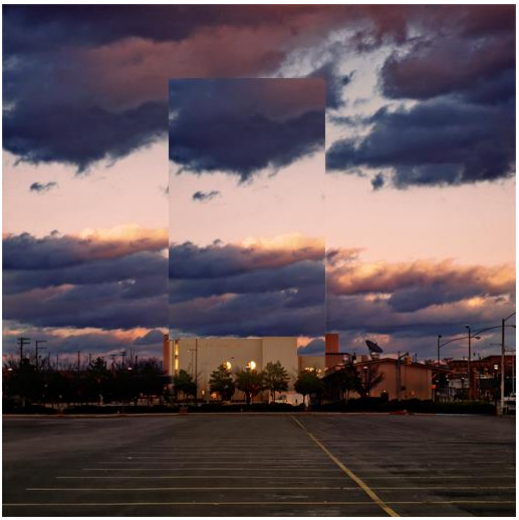


Gambar 5.20 *Ground truth* citra masukan 5 bernama 05_result.png

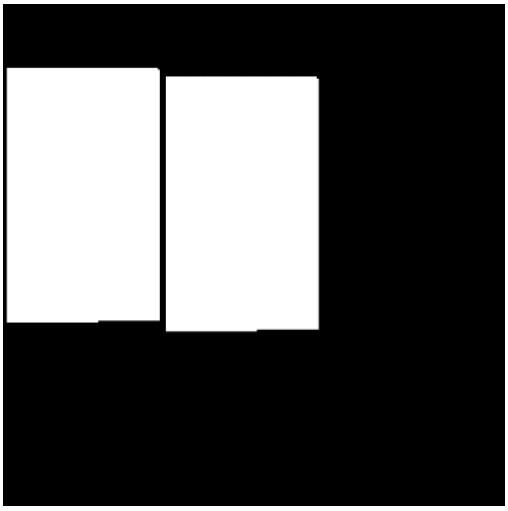
Gambar 5.21 Citra masukan 6 bernama 06_horses_copy.png



Gambar 5.22 *Ground truth* dari citra masukan 6 bernama 06_result.png



Gambar 5.23 Citra masukan 7 bernama 07_knight_moves_copy.png

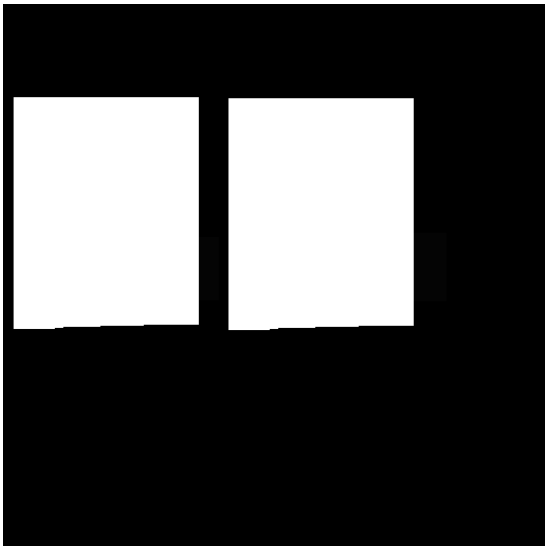


Gambar 5.24 *Ground truth* dari citra masukan 7 bernama 07_result.png

Gambar 5.25 Citra masukan 8 bernama 08_lone_cat_copy.png

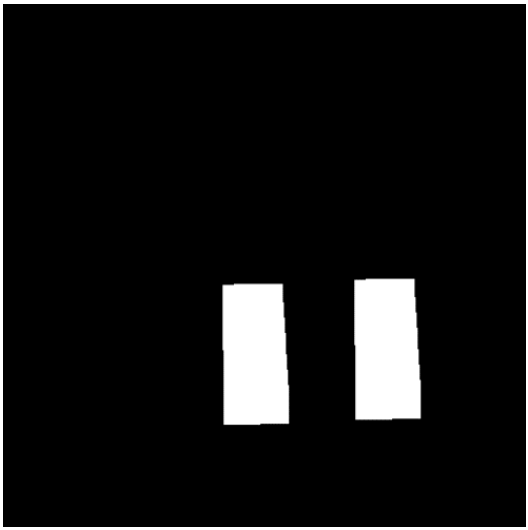
Gambar 5.26 *Ground truth* dari citra masukan 8 bernama 08_result.png

Gambar 5.27 Citra masukan 9 bernama 09_malawi_copy.png



Gambar 5.28 *Ground truth* dari citra masukan 9 bernama 09_result.png

Gambar 5.29 Citra masukan 10 dengan nama 10_mykene_copy.png



Gambar 5.30 *Ground truth* dari citra masukan 10 bernama 10_result.png

5.2.3 Citra yang diserang *copy-move* dan *blurring*

Subbab ini menjelaskan tentang citra yang dilakukan serangan *copy-move* lalu dilakukan proses *blurring*. Citra pada jenis data masukan ini mengambil citra pada subbab sebelumnya yakni citra yang hanya diserang *copy-move* saja, kemudian dilakukan proses *blurring* dengan menggunakan *blur tool* pada *Photoshop* agar tepi daerah yang dilakukan serangan *copy-move* dapat lebih samar sehingga data masukan lebih susah untuk dideteksi baik oleh algoritma maupun oleh mata manusia.

Karena daerah yang diduplikasi sama dengan citra yang hanya dilakukan serangan *copy-move* saja, maka *ground truth* dari jenis citra masukan ini adalah sama sama dengan *ground truth* dari jenis citra pada Subbab 5.2.2.

Pada Gambar 5.31 yang menjadi citra masukan 1, terlihat bahwa tepi daerah air laut yang diduplikasi menjadi samar, sehingga warna daerah yang diduplikasi menyatu dengan warna sekitarnya. Pada Gambar 5.32 yang menjadi citra masukan 2, terlihat bahwa tepi daerah rumput yang diduplikasi menjadi samar dan menyatu dengan rumput disekitarnya. Pada Gambar 5.33 yang menjadi citra masukan 3, terlihat bahwa tepi daerah tembok abu-abu yang diduplikasi menjadi samar dan menyatu dengan tembok disekitarnya. Pada Gambar 5.34 yang menjadi citra masukan 4, terlihat bahwa tepi daerah langit berawan yang diduplikasi menjadi samar dan menyatu dengan langit disekitarnya. Pada Gambar 5.35 yang menjadi citra masukan 5, terlihat bahwa tepi daerah handuk yang diduplikasi menjadi samar dan menyatu dengan daerah handuk diekitarnya.

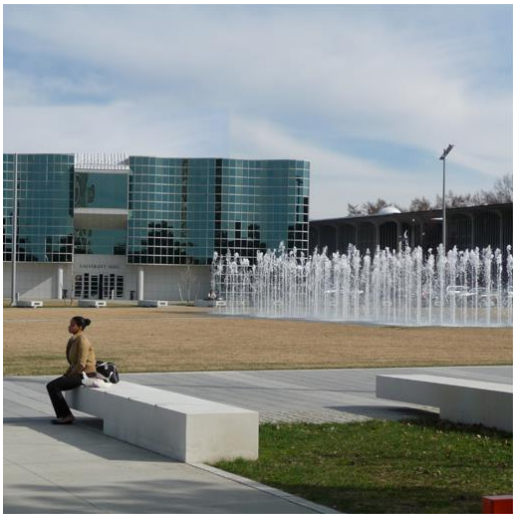
Pada Gambar 5.36 yang menjadi citra masukan 6, terlihat bahwa tepi daerah langit diduplikasi menjadi samar dan menyatu dengan langit disekitarnya. Pada Gambar 5.37 yang menjadi citra masukan 7, terlihat bahwa tepi daerah langit yang diduplikasi menjadi samar dan menyatu dengan daerah langit disekitarnya. Pada Gambar 5.38 yang menjadi citra masukan 8, terlihat bahwa tepi daerah alas yang diduplikasi menjadi samar dan menyatu

dengan daerah alas disekitarnya. Pada Gambar 5.39 yang menjadi citra masukan 9, terlihat bahwa tepi daerah langit yang diduplikasi menjadi samar dan menyatu dengan daerah langit disekitarnya. Pada Gambar 5.40 yang menjadi citra masukan 10, terlihat bahwa tepi daerah tembok bebatuan yang diduplikasi menjadi samar.

Gambar 5.31 Citra masukan 1 bernama 01_barrier_blur.png

Gambar 5.32 Citra masukan 2 bernama 02_cattle_blur.png

Gambar 5.33 Citra masukan 3 bernama 03_clean_walls_blur.png



Gambar 5.34 Citra masukan 4 bernama 04_fountain_blur.png

Gambar 5.35 Citra masukan 5 bernama 05_four_babies_blur.png

Gambar 5.36 Citra masukan 6 bernama 06_horses_blur.png

Gambar 5.37 Citra masukan 7 bernama 07_knight_moves_blur.png

Gambar 5.38 Citra masukan 8 bernama 08_lone_cat_blur.png

Gambar 5.39 Citra masukan 9 bernama 09_malawi_blur.png

Gambar 5.40 Citra masukan 10 bernama 10_mykene_blur.png

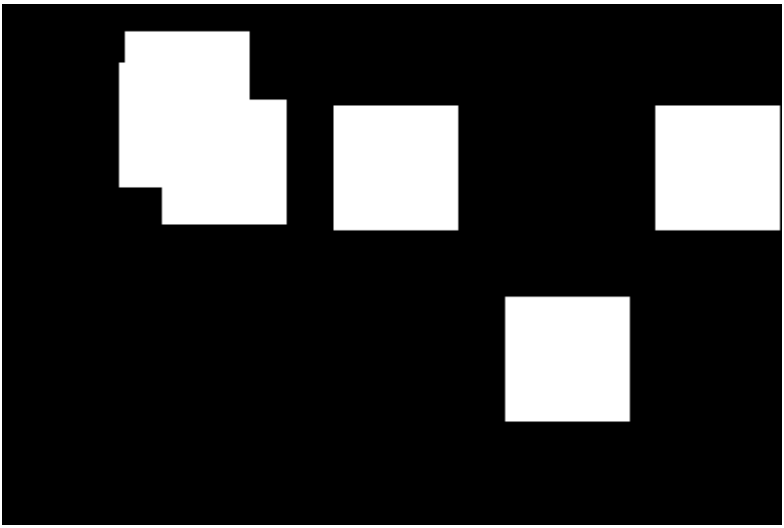
5.2.4 Citra dari *dataset* publik

Subbab ini menjelaskan tentang citra yang dilakukan serangan *copy-move* yang didapatkan dari *dataset* publik [21]. Citra pada jenis data masukan ini didapat dalam keadaan sudah dilakukan serangan *copy-move*.

Citra masukan pertama dan citra *ground truth*nya terdapat pada Gambar 5.41, dan Gambar 5.42. Citra ini memiliki kondis diluar batasan yakni daerah yang diduplikasi tertindih dengan daerah duplikasi yang lain.

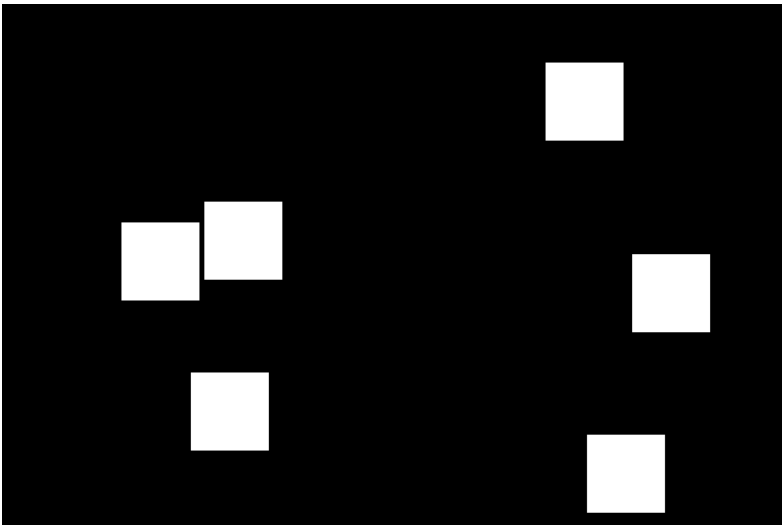
Citra masukan kedua dan citra *ground truth*nya terdapat pada Gambar 5.43, dan Gambar 5.44. Pada citra masukan ini tidak terdapat daerah duplikasi yang saling tertindih, sehingga sebenarnya memiliki kondisi seperti citra *copy-move* biasa yang telah dibuat pada Subbab 5.3.3.

Gambar 5.41 Citra masukan 1 bernama 01_giraffe.png



Gambar 5.42 Citra *ground truth* dari citra masukan 1 bernama 01_giraffe_answer.png

Gambar 5.43 Citra masukan 2 bernama 04_cattle.png



Gambar 5.44 Citra *ground truth* dari citra masukan 2 bernama 04_cattle_answer.png

5.3 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, serta membandingkan skenario manakah yang memiliki hasil paling baik. Terdapat 6 macam skenario uji coba, yaitu:

1. Penghitungan parameter terbaik untuk metode *duplication detection*, *robust detection*, dan *robust-duplication detection*.
2. Penghitungan akurasi pendeteksian *copy-move* terhadap citra otentik untuk mengetahui *false positive* menggunakan metode *duplication detection*, *robust detection*, dan *robust-duplication detection*.
3. Penghitungan akurasi pendeteksian *copy-move* terhadap citra yang diserang *copy-move* menggunakan metode

duplication detection, *robust detection*, dan *robust-duplication detection*.

4. Penghitungan akurasi pendeteksian *copy-move* yang disertai *blurring* pada citra dengan menggunakan metode *duplication detection*, *robust detection*, dan *robust-duplication detection*.
5. Penghitungan akurasi pendeteksian *copy-move* pada citra dari *dataset* publik dengan menggunakan metode *duplication detection*, *robust detection*, dan *robust-duplication detection*.
6. Perhitungan perbandingan akurasi pendeteksian *copy-move* terhadap citra yang diserang *copy-move* menggunakan metode *robust-duplication detection* pada penggunaan nilai parameter loncatan pembuatan *overlapping block* yang berbeda.

Seperti yang terdapat pada Subbab 5.2, *dataset* terdiri dari 30 buah citra yang dibuat dari 10 buah citra otentik yang didapat dari Universitas Friedich-Alexander yang memiliki lisensi MIT [21]. Lisensi ini menyatakan bahwa dataset dapat digunakan, di-*copy*, dimodifikasi, disatukan dengan data lain, dipublikasi, dan disebarkan dengan syarat menyertakan sumber dataset yang digunakan.

Dari citra otentik tersebut dilakukan serangan *copy-move* dengan menggunakan perangkat lunak *Photoshop*, dengan detail sebagai berikut:

1. 10 buah citra pertama merupakan citra otentik yang tidak dilakukan serangan sama sekali.
2. 10 buah citra yang dilakukan serangan *copy-move* biasa.
3. 10 buah citra yang dilakukan serangan *copy-move* dan dilakukan proses *post region duplication process*, dimana diambil salah satu proses yakni proses *blurring* yang dilakukan pada tepi daerah yang diduplikasi.

Masing-masing citra berukuran 512×512 piksel tersebut memiliki pasangan citra yang berlaku sebagai *ground truth* dari

serangan *copy-move*. Pada *dataset* berlaku asumsi bahwa duplikasi yang terjadi tidak boleh saling tertindih (*overlap*) serta untuk setiap citra hanya terdapat satu daerah yang dilakukan serangan duplikasi.

Selain *dataset* diatas, terdapat juga dua buah citra dari sumber yang sama dengan kondisi sudah dilakukan serangan *copy-move*. Kedua data citra ini mempresentasikan *dataset* publik.

Pada setiap skenario uji coba, akan dihitung nilai MSE, *similarity*, dan akurasi yang didapatkan dari *confusion matrix*. Nilai MSE diperoleh dari rataan kuadrat kesalahan dari citra hasil dengan citra *ground truth*. Semakin kecil nilai MSE maka semakin sedikit kesalahan yang ada antara citra hasil deteksi dengan *ground truth*. Nilai MSE ini memiliki rentang mulai dari nol hingga tak terhingga.

Nilai *similarity* didapatkan dari jumlah piksel dari citra hasil yang bernilai sama dengan piksel dari citra *ground truth* pada tempat yang bersesuaian, kemudian dibagi dengan jumlah piksel yang ada pada citra tersebut dan dikalikan 100%. Semakin besar nilai *similarity* (mendekati 100%), maka semakin mirip citra hasil deteksi dengan citra *ground truth*. Nilai *similarity* ini memiliki rentang dari nol hingga 100 dengan satuan persen.

Pada *confusion matrix*, terdapat empat nilai yakni *true positive* (TP), *true negative* (TN), *false positive* (FP), dan *false negative* (FN). *True positive* (TP) diambil dari jumlah pasangan daerah citra yang terdeteksi *copy-move* oleh metode dan memang daerah tersebut terduplikasi sesuai *ground truth*. *True negative* (TN) terjadi saat tidak ada satu pun daerah yang terdeteksi *copy-move* dan memang tidak ada daerah yang terduplikasi pada *ground truth*. *False positive* (FP) diambil dari jumlah pasangan citra yang terdeteksi *copy-move* oleh metode namun pada *ground truth* tidak terduplikasi, dengan kata lain bahwa metode melakukan kesalahan dugaan deteksi. *False negative* (FN) diambil dari jumlah pasangan blok citra pada *ground truth* yang tidak dapat terdeteksi *copy-move* oleh metode yang digunakan. Terakhir, nilai akurasi didapatkan dengan menjumlahkan *true positive* dan *true negative* dari *confusion matrix* kemudian dibagi dengan total data yang ada.

5.3.1 Skenario Uji Coba 1

Skenario uji coba 1 adalah penghitungan parameter terbaik pada metode *duplication detection*, *robust detection*, dan *robust-duplication detection*. Skenario dilakukan dengan memasukkan sepuluh citra *dataset* pada Subbab 5.2.2 dan menjalankan metode dengan berbagai kombinasi parameter.

Berbagai kombinasi parameter pada metode dan rataan MSE dari hasilnya terdapat pada Tabel 5.5. Telah diketahui dari Subbab sebelumnya bahwa parameter *Nn* mengatur seberapa jauh blok citra tetangga yang dicari pada pembuatan pasangan blok citra. Parameter *Nd* mengatur jarak *euclid* minimal antar pasangan blok citra untuk dapat dikatakan daerah *copy-move*. Parameter *Nf* mengatur frekuensi *offset* minimal dari kumpulan pasangan blok citra yang memiliki *offset* sama.

Pada percobaan 1, digunakan nilai parameter bawaan dari referensi jurnal yang digunakan yakni *Nn* sebesar 100, *Nd* sebesar 16, dan *Nf* sebesar 128. Parameter *Nn* pada percobaan 1 memiliki arti bahwa pencarian pasangan blok citra dilakukan sejauh 100 blok dari blok citra acuan. Parameter *Nd* dari percobaan 1 memiliki arti bahwa hanya pasangan blok citra yang memiliki jarak *euclid* lebih besar dari 16 yang akan dianggap sebagai daerah *copy-move*. Parameter *Nf* dari percobaan 1 memiliki arti bahwa harus terdapat minimal 128 buah pasangan blok citra dengan *offset* sama agar kumpulan blok citra tersebut bisa diakui sebagai daerah *copy-move*. Percobaan 1 menghasilkan citra yang mengandung banyak *false positive*, juga dibuktikan dengan nilai MSE yang sangat tinggi. Untuk saat ini, dugaan penyebab *false positive* dimungkinkan berasal dari parameter *Nn*, *Nd*, maupun *Nf* yang tidak optimal karena hanya mengacu pada referensi jurnal, sementara referensi jurnal menggunakan citra yang memiliki resolusi berbeda sebagai bahan uji coba. Untuk itu, percobaan harus dilanjutkan dengan kombinasi lain, dengan kesimpulan sementara bahwa parameter pada percobaan 1 tidak dapat diguakan untuk *dataset* pada tugas akhir ini.

Pada percobaan 2, dilakukan pengurangan nilai *Nn* menjadi 50, yang berarti bahwa metode akan mencari pasangan blok citra sejauh 50 blok tetangga dari citra acuan. Hasil percobaan 2 menunjukkan nilai rataan MSE yang sama. Dari sini didapatkan kesimpulan sementara bahwa parameter *Nn* yang terlalu besar tidak akan mempengaruhi citra dengan ukuran yang relatif kecil. Parameter *Nn* dapat bernilai besar (misal ribuan), hanya saja hal tersebut hanya akan menambah kemungkinan pasangan blok citra yang terbuat. Sementara di akhir tahapan metode, semua pasangan blok citra yang tidak sesuai dengan parameter *Nd* dan *Nf* tetap tidak akan digunakan.

Pada percobaan 3, dilakukan lagi pengurangan nilai *Nn* menjadi 2, yang berarti bahwa metode akan mencari pasangan blok citra sejauh 2 blok tetangga dari citra acuan. Hasil percobaan 3 memberikan nilai rataan MSE yang masih sama. Dari sini didapatkan kesimpulan bahwa nilai parameter *Nn* akan berbanding lurus dengan ukuran citra. Semakin besar citra, maka semakin banyak *overlapping block* yang ada, membuat semakin panjang matriks penampung blok citra, sehingga metode harus mencari pasangan blok citra dengan jarak ketetanggaan yang lebih jauh pula. Referensi jurnal menggunakan *Nn* sebesar 100, karena citra pada uji coba yang dilakukan memiliki resolusi besar. Sehingga parameter *Nn* sebesar 2 dapat dikatakan cukup untuk digunakan pada *dataset* citra tugas akhir ini yang relatif lebih kecil.

Pada percobaan 4, dilakukan penambahan nilai *Nd* menjadi 50. Hal ini memiliki arti bahwa metode akan mencari pasangan blok citra yang minimal memiliki jarak euclid sebesar 50, dan sisanya tidak akan digunakan. Percobaan 4 memberikan hasil yang lebih baik dengan turunnya nilai rataan MSE. Parameter *Nd* yang terlalu kecil akan mengakibatkan munculnya daerah deteksi yang berjarak sangat dekat. Maka peningkatan nilai parameter *Nd* akan berperan penting dalam menjaga agar blok citra berwarna homogen yang saling berdekatan (seperti rerumputan atau langit) tidak akan terdeteksi *copy-move*.

Pada percobaan 5, dilakukan lagi penambahan nilai *Nd* menjadi 100, yang berarti metode hanya akan mendeteksi pasangan blok citra yang memiliki jarak *euclid* minimal sebesar 100. Ternyata percobaan 5 memberikan nilai rataan MSE yang menurun, namun turunnya rataan MSE tersebut disebabkan karena nilai *Nd* yang terlalu besar sehingga bahkan daerah yang sebenarnya merupakan *copy-move* menjadi tidak terdeteksi karena jaraknya terlalu kecil jika dibandingkan parameter *Nd* yang digunakan sehingga metode tidak dapat mendeteksi daerah *copy-move* sama sekali. Hasil pada percobaan 5 ini adalah *false negative* untuk semua citra, dimana metode mengatakan bahwa citra masukan adalah otentik padahal semuanya memiliki daerah *copy-move*. Maka dari percobaan 4 dan 5 ini didapatkan kesimpulan bahwa parameter *Nd* yang efektif digunakan untuk *dataset* citra pada tugas akhir ini adalah sebesar 50.

Pada percobaan 6, dilakukan penambahan nilai parameter *Nf* menjadi 200, yang memiliki arti bahwa metode hanya akan mendeteksi daerah *copy-move* yang memiliki frekuensi *offset* diatas 200. Ternyata percobaan 6 memberikan hasil yang lebih baik dengan turunnya nilai MSE. Hal ini disebabkan karena naiknya nilai parameter *Nf*, membuat metode lebih ketat dalam pemutusan daerah *copy-move*. Pada blok citra otentik yang memiliki warna homogen, tentu akan memiliki fitur karakteristik yang sama. Namun, frekuensi *offset* blok citra tersebut akan jauh lebih kecil jika dibandingkan blok citra yang memang disebabkan *copy-move*, sesuai asumsi yang digunakan pada Subbab 1.3 bahwa daerah *copy-move* harus minimal berukuran 0.85% dari citra total. Didapatkan kesimpulan sementara bahwa penambahan nilai parameter *Nf* dapat lebih merapikan hasil deteksi karena dapat menghilangkan blok citra yang memiliki frekuensi *offset* kecil.

Pada percobaan 7 hingga 9, dilakukan penambahan nilai parameter *Nf* menjadi 300, 500, dan 750 dengan maksud seperti pada percobaan 6. Seiring dengan meningkatnya nilai parameter *Nf* maka rataan MSE yang dihasilkan juga semakin kecil, yang berarti hasil semakin membaik. Hingga pada akhir percobaan 9,

didapatkan MSE terbaik yakni sebesar 580.10. Citra hasil deteksi pada percobaan 9 ini relatif lebih rapi, dan sudah tidak ada *false positive* yang terjadi.

Pada percobaan 10, dilakukan penambahan nilai parameter *Nf* menjadi 1000, namun justru memberikan hasil *false negative* untuk semua citra, dibuktikan dengan nilai rataan MSE yang sama dengan percobaan 5. Didapatkan kesimpulan sementara bahwa nilai 1000 pada parameter *Nf* sudah terlalu besar, sehingga bahkan daerah *copy-move* saja tidak memiliki frekuensi *offset* yang lebih besar dari nilai parameter yang diberikan, mengakibatkan tidak ada satupun blok citra yang dapat terdeteksi *copy-move* oleh metode karena tidak ada blok citra yang memiliki frekuensi *offset* melebihi 1000.

Tabel 5.5 Berbagai kombinasi parameter metode dan rataan MSE yang dihasilkan

Percobaan	**Parameter**			**Rataan MSE**
	Nn	*Nd*	*Nf*	
1	100	16	128	153872.25
2	50	16	128	153872.25
3	2	16	128	153872.25
4	2	50	128	8544.58
5	2	100	128	2072.24
6	2	50	200	7563.52
7	2	50	300	6935.39
8	2	50	500	610.32
9	**2**	**50**	**750**	**580.10**
10	2	50	1000	2072.24

Untuk parameter khusus seperti *t1*, *t2*, dan *P* yang berkaitan dengan kepekaan metode dalam menangani perbedaan fitur karakteristik maka digunakan nilai dari referensi jurnal,

dengan *t1* sebesar 2.80, *t2* sebesar 0.02, dan matriks *P* dengan nilai [1.8, 1.8, 1.8, 0.0125, 0.0125, 0.0125, 0.0125].

Berdasarkan percobaan 1 hingga 10, maka didapatkan nilai parameter yang paling optimal adalah *Nn* sebesar 2, *Nd* sebesar 50 dan *Nf* sebesar 750 yang digunakan pada percobaan 9. Kombinasi nilai parameter ini kemudian digunakan untuk mendapatkan hasil pada skenario uji coba selanjutnya.

5.3.2 Skenario Uji Coba 2

Skenario uji coba 2 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *duplication detection*, *robust detection*, dan modifikasi *robust-duplication detection* pada citra yang otentik. Skenario dilakukan dengan memasukkan citra otentik kedalam tiga metode tersebut.

Nilai MSE, *similarity*, dan *confusion matrix* setiap metode pada hasil uji coba 2 terdapat pada Tabel 5.6, Tabel 5.7, Tabel 5.8, Tabel 5.9, dan Tabel 5.10. Berdasarkan ketiga hasil tersebut, baik metode *duplication detection*, *robust detection*, dan *robust-duplication detection* memiliki akurasi tinggi yakni mencapai **100%**. Hal ini memiliki arti bahwa metode *duplication detection*, *robust detection*, dan gabungan kedua metode tersebut yakni *robust-duplication detection* dapat dengan benar memberikan hasil bahwa semua citra masukan adalah otentik dan tidak terdapat daerah yang diduga diserang *copy-move*.

Contoh citra data masukan pada uji coba 2 ada pada Gambar 5.45, sedangkan *ground truth* dari citra tersebut ada pada Gambar 5.46. Hasil uji coba 2 pada citra masukan pada gambar Gambar 5.45 menggunakan *duplication detection* terdapat pada Gambar 5.47, menggunakan *robust detection* terdapat pada Gambar 5.48, dan menggunakan *robust-duplication detection* terdapat pada Gambar 5.49. Semua citra hasil uji coba 1 memiliki warna hitam secara keseluruhan, menandakan bahwa tidak ada kesalahan dugaan (*false positive*) yang terjadi.

Hasil akhir uji coba 2 pada Gambar 5.50 , merupakan hasil akhir yang didapat dari metode *robust-detection algorithm*. Daerah yang diduga identik karena searangan *copy-move* akan diberi garis berwarna hijau. Karena tidak ada daerah yang terduga diserang *copy-move*, maka citra hasil akhir identik dengan citra masukan.

Skenario uji coba 2 ini memberikan hasil bahwa ketiga metode yang digunakan yakni *duplication detection*, *robust detection*, dan *robust-duplication detection* memiliki akurasi yang sama baiknya dalam hal pembuktian keotentikan sebuah citra.

Tabel 5.6 Nilai MSE hasil uji coba 2 terhadap *ground truth*

Nama file	***duplication detection***	***robust detection***	***robust-duplication detection***
01_barrier	0	0	0
02_cattle	0	0	0
03_clean_walls	0	0	0
04_fountain	0	0	0
05_four_babies	0	0	0
06_horses	0	0	0
07_knight_move	0	0	0
08_lone_cat	0	0	0
09_malawi	0	0	0
10_mykene	0	0	0

Tabel 5.7 Nilai *similarity* dari hasil uji coba 2 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_barrier	100.00%	100.00%	100.00%
02_cattle	100.00%	100.00%	100.00%
03_clean_walls	100.00%	100.00%	100.00%
04_fountain	100.00%	100.00%	100.00%
05_four_babies	100.00%	100.00%	100.00%
06_horses	100.00%	100.00%	100.00%
07_knight_moves	100.00%	100.00%	100.00%
08_lone_cat	100.00%	100.00%	100.00%
09_malawi	100.00%	100.00%	100.00%
10_mykene	100.00%	100.00%	100.00%
Rata-rata	**100.00%**	**100.00%**	**100.00%**

Tabel 5.8 *Confusion matrix* pada hasil uji coba 2 menggunakan metode *duplication detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier.png	0	0	0	1	100%
02_cattle.png	0	0	0	1	100%
03_clean_walls.png	0	0	0	1	100%
04_fountain.png	0	0	0	1	100%
05_four_babies.png	0	0	0	1	100%
06_horses.png	0	0	0	1	100%
07_knight_moves.png	0	0	0	1	100%
08_lone_cat.png	0	0	0	1	100%
09_malawi_png	0	0	0	1	100%
10_mykene.png	0	0	0	1	100%

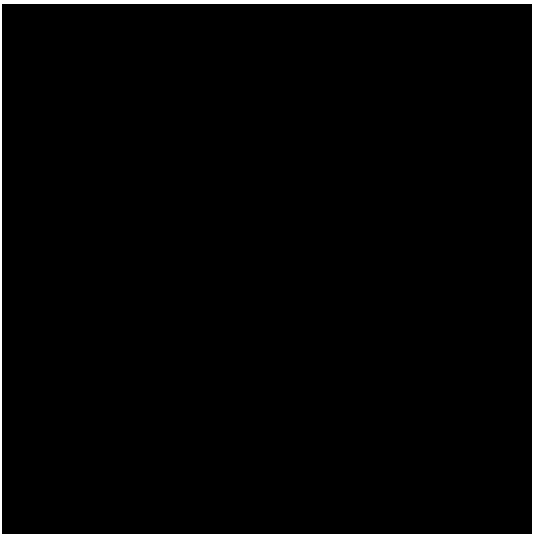
Tabel 5.9 *Confusion matrix* pada hasil uji coba 2 menggunakan metode *robust detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier.png	0	0	0	1	100%
02_cattle.png	0	0	0	1	100%
03_clean_walls.png	0	0	0	1	100%
04_fountain.png	0	0	0	1	100%
05_four_babies.png	0	0	0	1	100%
06_horses.png	0	0	0	1	100%
07_knight_moves.png	0	0	0	1	100%
08_lone_cat.png	0	0	0	1	100%
09_malawi_png	0	0	0	1	100%
10_mykene.png	0	0	0	1	100%

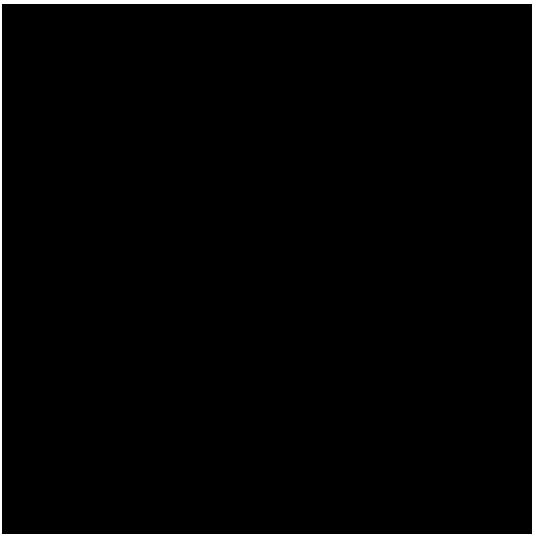
Tabel 5.10 *Confusion matrix* pada hasil uji coba 2 menggunakan metode *robust-duplication detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier.png	0	0	0	1	100%
02_cattle.png	0	0	0	1	100%
03_clean_walls.png	0	0	0	1	100%
04_fountain.png	0	0	0	1	100%
05_four_babies.png	0	0	0	1	100%
06_horses.png	0	0	0	1	100%
07_knight_moves.png	0	0	0	1	100%
08_lone_cat.png	0	0	0	1	100%
09_malawi_png	0	0	0	1	100%
10_mykene.png	0	0	0	1	100%

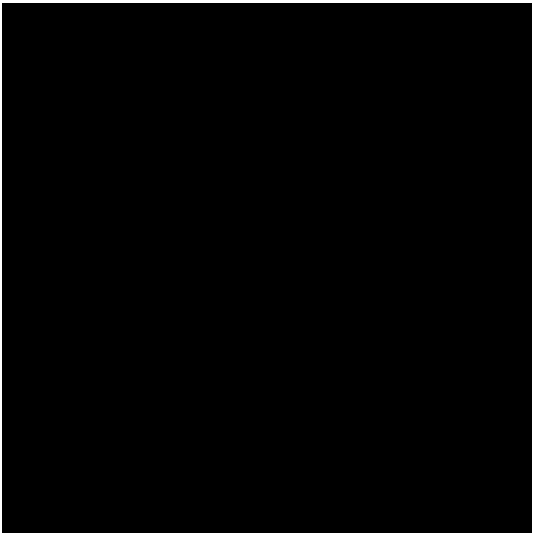
Gambar 5.45 Contoh citra masukan pada uji coba 2



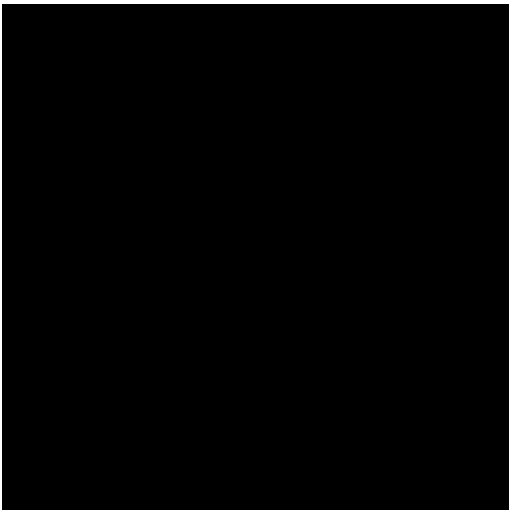
Gambar 5.46 Citra *ground truth* pada uji coba 2



Gambar 5.47 Hasil uji coba 2 menggunakan *duplication detection*



Gambar 5.48 Hasil uji coba 2 menggunakan *robust detection*



Gambar 5.49 Hasil uji coba 2 menggunakan *robust-duplication detection*

Gambar 5.50 Hasil akhir uji coba 2 dengan *robust-duplication detection*

5.3.3 Skenario Uji Coba 3

Skenario uji coba 3 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *duplication detection*, *robust detection*, dan modifikasi *duplication detection-robust detection* pada citra yang dilakukan serangan *copy-move* biasa. Skenario dilakukan dengan memasukkan citra yang diserang *copy-move* kedalam tiga metode tersebut.

Nilai MSE, *similarity*, dan *confusion matrix* setiap metode pada hasil uji coba 3 terdapat pada Tabel 5.11, Tabel 5.12, Tabel 5.13, Tabel 5.14, dan Tabel 5.15. Berdasarkan nilai yang ditunjukkan pada beberapa tabel hasil, citra masukan yang diserang *copy-move* saja akan sangat efektif jika dilakukan pendeteksian dengan metode *duplication detection* dengan rata-rata MSE sebesar **542.40**, sementara metode *robust detection* memiliki rata-rata MSE hingga **1023.98**.

Unggulnya metode *duplication detection* pada kasus ini disebabkan karena metode *principal component analysis* yang digunakan oleh algoritma *duplication detection* dapat secara akurat memberikan fitur karakteristik blok citra. Keunggulan tersebut mengakibatkan hasil daerah teruduga diserang *copy-move* memiliki tepi yang halus, berujung kepada kecilnya MSE yang dihasilkan. Berhubung citra masukan tidak dilakukan *post region duplication process* seperti *blurring*, maka algoritma akan dengan mudah membandingkan blok citra yang identik berdasarkan *principal component* yang dijadikan sebagai fitur karakteristik tersebut. Namun dalam hal akurasi, penggunaan *principal component* mengakibatkan blok citra dengan warna homogen akan dapat ikut terduga merupakan hasil serangan *copy-move*, mengakibatkan terdapat tiga dari sepuluh citra masukan terjadi kesalahan deteksi (*false positive*) atau memiliki daerah terduga diserang *copy-move* yang sebenarnya daerah citra tersebut adalah otentik. Contoh keluaran kesalahan deteksi tersebut terdapat pada Gambar 5.51 yang salah pada citra tembok berwarna cokelat,

Gambar 5.52 yang salah pada citra langit serta pada dinding, dan Gambar 5.53 yang salah pada citra langit berawan.

Tabel 5.11 Nilai MSE dari hasil uji coba 3 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_barrier	0.00	253.01	250.04
02_cattle	0.00	409.28	410.77
03_clean_walls	1524.02	47.63	80.37
04_fountain	0.00	514.95	190.50
05_four_babies	0.00	695.04	235.15
06_horses	3048.05	445.00	117.58
07_knight_move	762.01	1713.04	1704.11
08_lone_cat	43.16	692.06	870.66
09_malawi	46.71	3939.82	1935.01
10_mykene	0.00	1529.98	5.95
Rata-rata	**542.40**	**1023.98**	**580.01**

Pada algoritma *robust detection*, fitur karakteristik yang diambil adalah berdasarkan perbandingan nilai piksel. Dalam keadaan citra masukan normal, algoritma ini akan selalu lebih buruk dibanding algoritma pertama dengan memiliki MSE yang jauh lebih besar. Hal ini disebabkan karena aturan perbandingan piksel yang digunakan sebagai fitur karakteristik pada metode ini tidak secara keseluruhan merepresentasikan blok citra. Kelemahan tersebut dirasakan pada kasarnya tepi daerah hasil deteksi yang disebabkan karena fitur karakteristik yang tidak dapat merepresentasikan semua blok citra yang ada, sehingga terdapat beberapa blok citra yang sebenarnya termasuk daerah hasil serangan *copy-move* namun tidak terdeteksi, terutama pada tepi daerah yang diduplikasi. Contoh keluaran yang memiliki tepi kasar

terdapat pada Gambar 5.54, Gambar 5.55, dan Gambar 5.56. Pada sisi keunggulannya, algoritma ini tidak mudah mengalami kesalahan deteksi pada blok citra yang memang otentik, dibuktikan dengan hanya terdapat dua dari sepuluh citra yang mengalami kesalahan deteksi (*false positive*). Hal ini terjadi karena alasan yang sama dengan penjelasan sebelumnya, yakni fitur karakteristik yang tidak terlalu merepresentasikan blok citra secara keseluruhan, sehingga blok citra yang memiliki warna identikpun dapat dihiraukan oleh algoritma.

Tabel 5.12 Nilai *smilarity* dari hasil uji coba 3 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_barrier	100.00%	99.87%	99.87%
02_cattle	100.00%	99.79%	99.79%
03_clean_walls	99.22%	99.98%	99.96%
04_fountain	100.00%	99.74%	99.90%
05_four_babies	100.00%	99.64%	99.88%
06_horses	98.44%	99.77%	99.94%
07_knight_move	99.61%	99.12%	99.13%
08_lone_cat	99.98%	99.65%	99.55%
09_malawi	98.80%	96.96%	97.85%
10_mykene	100.00%	99.22%	100.00%
Rata-rata	**99.60%**	**99.37%**	**99.59%**

Algoritma gabungan kedua metode yakni *robust-duplication detection* mampu beradaptasi terhadap citra masukan. Fungsi pembulatan (*round*) yang terdapat pada algoritma tersebut memberikan toleransi kepada *duplication detection* yang diyakini lebih handal untuk dapat mendominasi proses deteksi pada kasus serangan *copy-move* normal, mengakibatkan metode ini memiliki

rata-rata MSE yang mendekati *duplication detection* yakni sebesar **580.1**. Metode ini menghasilkan dugaan daerah duplikasi yang lebih halus dibandingkan hasil dari metode *robust detection* karena menggunakan *principal component* dari metode *duplication detection* sebagai salah satu fitur karakteristiknya. Metode ini juga memiliki akurasi yang baik, yakni **100%** dengan arti bahwa tidak ada kesalahan yang terjadi (baik *error* tipe 1 maupun *error* tipe 2) pada sepuluh citra masukan. Hal ini dapat dicapai karena menggunakan aturan perbandingan nilai piksel dari metode *robust detection* sebagai fitur karakteristik pendukung. Contoh keluaran dari algoritma *robust-duplication detection* dapat dilihat pada Gambar 5.57, Gambar 5.58, dan Gambar 5.59.

Tabel 5.13 *Confusion matrix* pada hasil uji coba 3 menggunakan metode *duplication detection*

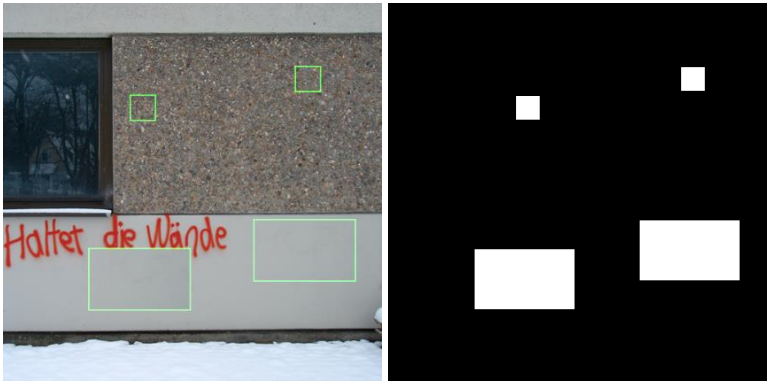
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_copy.png	1	0	0	0	100%
02_cattle_copy.png	1	0	0	0	100%
03_clean_walls_copy.png	1	1	0	0	50%
04_fountain_copy.png	1	0	0	0	100%
05_four_babies_copy.png	1	0	0	0	100%
06_horses_copy.png	1	2	0	0	33%
07_knight_moves_copy	1	1	0	0	50%
08_lone_cat_copy.png	1	0	0	0	100%
09_malawi_copy_png	1	0	0	0	100%
10_mykene_copy.png	1	0	0	0	100%

Tabel 5.14 *Confusion matrix* pada hasil uji coba 3 menggunakan metode *robust detection*

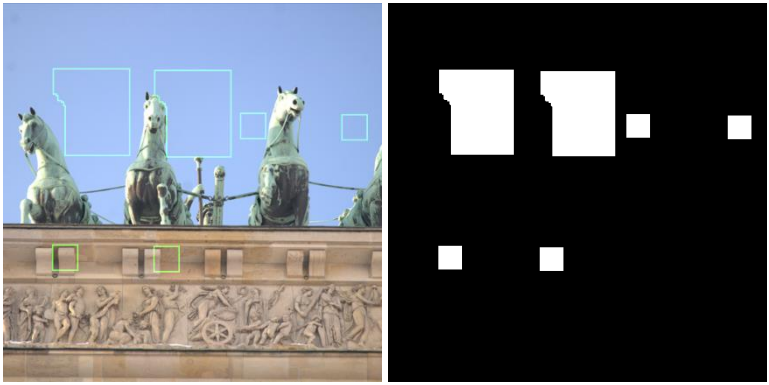
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_copy.png	1	0	0	0	100%
02_cattle_copy.png	1	0	0	0	100%
03_clean_walls_copy.png	1	0	0	0	100%
04_fountain_copy.png	1	0	0	0	100%
05_four_babies_copy.png	1	0	0	0	100%
06_horses_copy.png	1	0	0	0	100%
07_knight_moves_copy	1	0	0	0	100%
08_lone_cat_copy.png	1	0	0	0	100%
09_malawi_copy_png	1	1	0	0	50%
10_mykene_copy.png	1	1	0	0	50%

Tabel 5.15 *Confusion matrix* pada hasil uji coba 3 menggunakan metode *robust-duplication detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_copy.png	1	0	0	0	100%
02_cattle_copy.png	1	0	0	0	100%
03_clean_walls_copy.png	1	0	0	0	100%
04_fountain_copy.png	1	0	0	0	100%
05_four_babies_copy.png	1	0	0	0	100%
06_horses_copy.png	1	0	0	0	100%
07_knight_moves_copy	1	0	0	0	100%
08_lone_cat_copy.png	1	0	0	0	100%
09_malawi_copy_png	1	0	0	0	100%
10_mykene_copy.png	1	0	0	0	100%

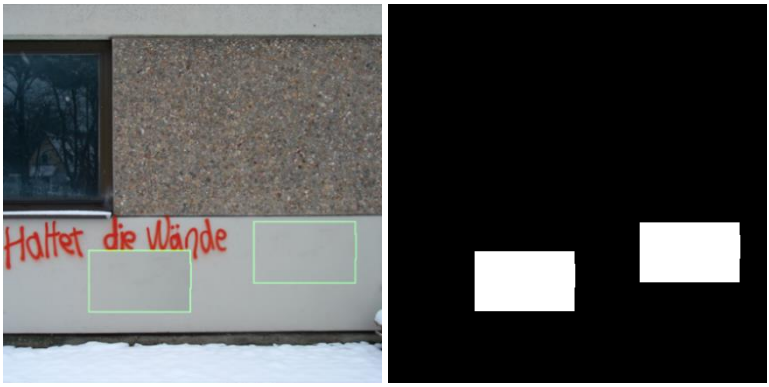


Gambar 5.51 Hasil deteksi dan hasil akhir citra masukan 3 dengan metode *duplication detection*

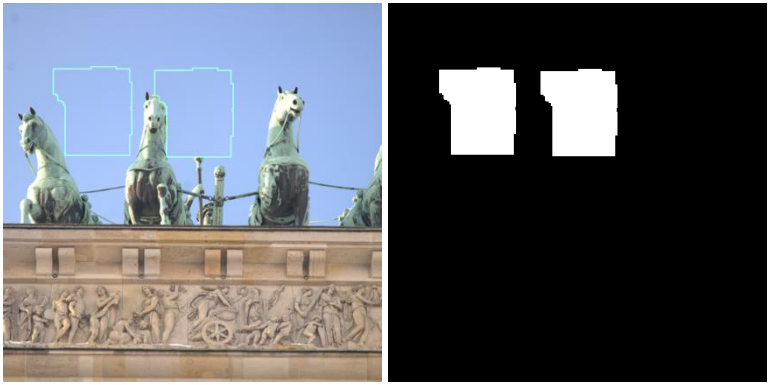


Gambar 5.52 Hasil deteksi dan hasil akhir citra masukan 6 dengan metode *duplication detection*

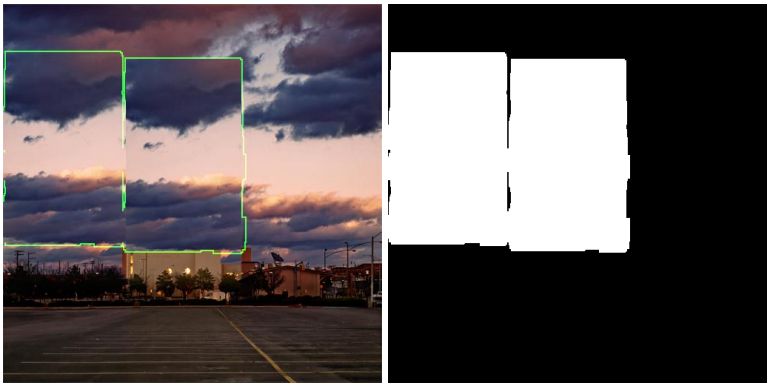
Gambar 5.53 Hasil deteksi dan hasil akhir citra masukan 7 dengan metode *duplication detection*



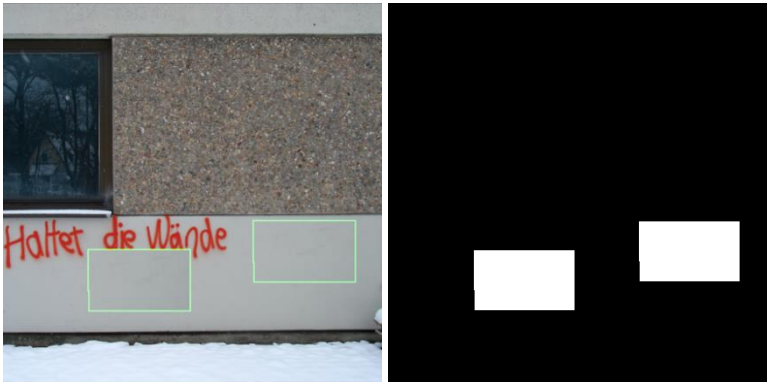
Gambar 5.54 Hasil deteksi dan hasil akhir citra masukan 3 dengan metode *robust detection*



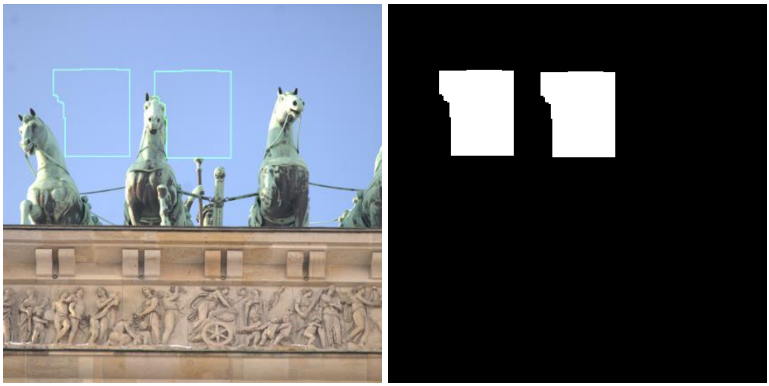
Gambar 5.55 Hasil deteksi dan hasil akhir citra masukan 6 dengan metode *robust detection*



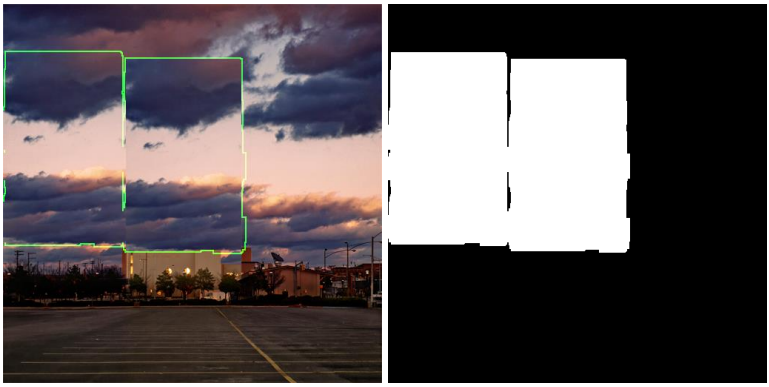
Gambar 5.56 Hasil deteksi dan hasil akhir citra masukan 7 dengan metode *robust detection*



Gambar 5.57 Hasil deteksi dan hasil akhir citra masukan 3 dengan metode *robust-duplication detection*



Gambar 5.58 Hasil deteksi dan hasil akhir citra masukan 6 dengan metode *robust-duplication detection*



Gambar 5.59 Hasil deeksi dan hasil akhir citra masukan 7 dengan metode *robust-duplication detection*

5.3.4 Skenario Uji Coba 4

Skenario uji coba 4 adalah penghitungan akurasi pendeteksian *copy-move* citra dengan menggunakan metode *duplication detection*, *robust detection*, dan modifikasi *duplication detection-robust detection* pada citra yang dilakukan serangan *copy-move* dan *blurring*. Skenario dilakukan dengan memasukkan citra yang diserang *copy-move* dilanjutkan serangan *blurring* kedalam tiga metode tersebut.

Nilai MSE, *similarity*, dan *confusion matrix* setiap metode pada hasil uji coba 4 terdapat pada Tabel 5.16, Tabel 5.17, Tabel 5.18, Tabel 5.19, dan Tabel 5.20. Berdasarkan nilai yang ditunjukkan pada beberapa tabel hasil, citra masukan yang diserang *copy-move* dan *blurring* akan efektif jika dilakukan pendeteksian dengan metode *robust detection* dengan rata-rata MSE sebesar **2538.93** dibandingkan menggunakan metode *duplication detection* yang memiliki rata-rata MSE hingga **7504.83**.

Pada algoritma *duplication detection*, fitur karakteristik diambil menggunakan *principal component analysis* dari keseluruhan blok citra, yang berarti blok citra yang dilakukan *blurring* akan memiliki *principal component* yang berbeda jauh

dengan blok citra yang tidak dilakukan *blurring*. Perbedaan pada *principal component* tersebut mengakibatkan tidak terdeteksinya bagian blok citra yang telah dilakukan *blurring*, sehingga metode hanya dapat mendeteksi daerah citra yang tidak dilakukan *blurring* saja. Jika intensitas proses *blurring* ditingkatkan dengan melakukan *blur* di seluruh daerah yang dilakukan *copy-move*, maka metode ini tidak akan dapat mendeteksi serangan sama sekali.

Tabel 5.16 Nilai MSE dari hasil uji coba 4 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_barrier	4929.26	1064.14	901.91
02_cattle	5971.08	1878.24	2073.21
03_clean_walls	6715.23	1899.08	1439.19
04_fountain	4064.56	894.47	878.10
05_four_babies	11239.67	4247.62	3443.94
06_horses	10013.31	4475.33	3970.80
07_knight_move	22266.52	3040.61	3001.17
08_lone_cat	7123.02	985.26	980.79
09_malawi	46.71	3939.82	1935.01
10_mykene	2678.95	2964.70	1404.96
Rata-rata	**7504.83**	**2538.93**	**2002.91**

Contoh hasil keluaran metode terdapat pada Gambar 5.60, Gambar 5.61, dan Gambar 5.62. Pada ketiga gambar tersebut terlihat bahwa hanya daerah yang tidak dilakukan *blurring* saja yang terdeteksi oleh metode ini. Meski pada skenario uji coba sebelumnya metode ini menunjukkan hasil yang baik, kekurangan metode pada citra masukan yang dilakukan *blurring* membuat metode mengeluarkan hasil yang kasar. Terlebih lagi kelemahan

metode ini pada skenario uji coba sebelumnya yakni mudah terjadi *false positive*, terbukti dengan adanya beberapa daerah pada gambar hasil yang terdeteksi serangan *copy-move* pada bagian yang sebenarnya otentik.

Tabel 5.17 Nilai *smilarity* dari hasil uji coba 4 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_barrier	97.47%	99.45%	99.54%
02_cattle	96.94%	99.04%	98.94%
03_clean_walls	96.56%	99.03%	99.26%
04_fountain	97.92%	99.54%	99.55%
05_four_babies	94.24%	97.82%	98.23%
06_horses	94.87%	97.71%	97.96%
07_knight_move	88.59%	98.44%	98.46%
08_lone_cat	96.35%	99.49%	99.50%
09_malawi	98.80%	96.96%	97.85%
10_mykene	98.63%	98.48%	99.28%
Rata-rata	**96.03%**	**98.60%**	**98.86%**

Unggulnya metode *robust detection* pada kasus ini disebabkan karena aturan perbandingan nilai piksel yang digunakan oleh metode untuk mendapatkan fitur karakteristik memiliki kehandalan terhadap perubahan piksel yang disebabkan oleh *post region duplication process*, seperti proses *blurring*. Kehandalan ini dapat terjadi karena fitur tersebut mewakili perbandingan jumlah nilai piksel pada suatu pola tertentu. Perbandingan jumlah nilai piksel ini tidak akan berubah jauh meskipun dilakukan proses *blurring*, karena sejatinya proses *blurring* hanya meratakan nilai piksel tanpa merubah jumlah nilai piksel pada suatu daerah. Penggunaan fitur tersebut membuat

metode ini dapat mengenali blok citra yang identik meski salah satu blok memiliki nilai piksel berbeda akibat dilakukan proses *blurring*. Contoh hasil keluaran metode terdapat pada Gambar 5.63, Gambar 5.64, dan Gambar 5.65. Pada ketiga gambar tersebut terlihat bahwa bagian citra yang samar akibat dilakukan *blurring* tetap ikut masuk kedalam daerah yang terduga dilakukan serangan *copy-move*. Metode ini tetap memiliki kelemahan yang sama yakni hasil daerah yang diduga terdeteksi tidak berbentuk halus, meskipun serangan *copy-move* dilakukan pada daerah yang berbentuk persegi dengan tepi yang halus.

Tabel 5.18 *Confusion matrix* pada hasil uji coba 4 menggunakan metode *duplication detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_blur.png	1	0	0	0	100%
02_cattle_blur.png	1	0	0	0	100%
03_clean_walls_blur.copy	1	1	0	0	50%
04_fountain_blur.png	1	0	0	0	100%
05_four_babies_blur.png	0	0	1	0	0%
06_horses_blur.png	0	0	1	0	0%
07_knight_moves_blur.png	1	0	0	0	100%
08_lone_cat_blur.png	1	0	0	0	100%
09_malawi_blur.png	1	0	0	0	100%
10_mykene_blur.png	1	0	0	0	100%

Algoritma gabungan kedua metode yakni *robust-duplication detection* mampu beradaptasi terhadap citra masukan. Fungsi pembulatan (*round*) yang terdapat pada algoritma tersebut memberikan toleransi kepada *robust detection* yang diyakini lebih handal untuk dapat mendominasi proses deteksi pada kasus serangan *copy-move* yang disertai *blurring*, mengakibatkan metode ini memiliki kehandalan dari metode *robust detection* untuk

mendeteksi serangan *copy-move* pada daerah yang dilakukukan *blurring* sekaligus keakuratan pada metode *duplication detection* dalam mendeteksi *copy-move* pada citra biasa yang tidak dilakukan *blurring*. Metode ini memiliki rata-rata MSE yang paling baik dibandingkan kedua metode sebelumnya yakni sebesar **2002.91**.

Tabel 5.19 *Confusion matrix* pada hasil uji coba 4 menggunakan metode *robust detection*

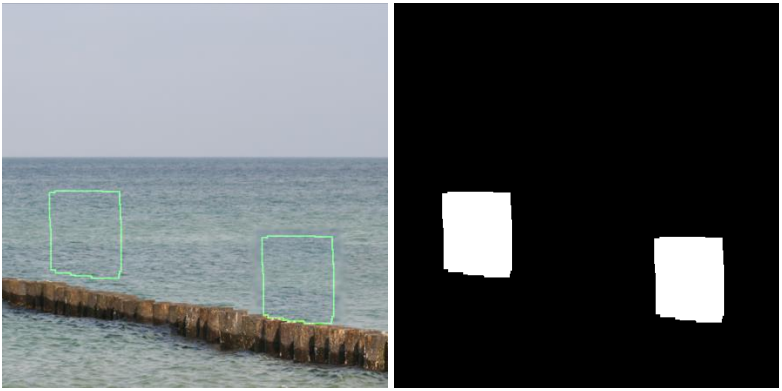
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_blur.png	1	0	0	0	100%
02_cattle_blur.png	1	0	0	0	100%
03_clean_walls_blur.copy	1	0	0	0	100%
04_fountain_blur.png	1	0	0	0	100%
05_four_babies_blur.png	0	0	1	0	0%
06_horses_blur.png	1	0	0	0	100%
07_knight_moves_blur.png	1	0	0	0	100%
08_lone_cat_blur.png	1	0	0	0	100%
09_malawi_blur.png	1	1	0	0	50%
10_mykene_blur.png	1	1	0	0	50%

Metode ini menghasilkan jawaban daerah duplikasi yang lebih halus terutama pada daerah tepi yang dilakukan *blurring* karena menggunakan aturan perbandingan nilai piksel dari metode *robust detection* yang handal terhadap *blurring* sebagai fitur dominannya. Pada bagian citra yang tidak *blur*, maka fungsi pembulatan akan membuat proses deteksi diambil alih oleh metode *duplication detection* sehingga memberikan hasil yang akurat. Metode ini juga memiliki akurasi yang baik yakni **100%** dengan arti bahwa tidak ada kesalahan yang terjadi (baik *error* tipe 1 maupun *error* tipe 2) pada sepuluh citra masukan. Hal ini dapat dicapai karena toleransi metode *duplication detection* terhadap *robust detection* dengan memenangkan aturan perbandingan nilai

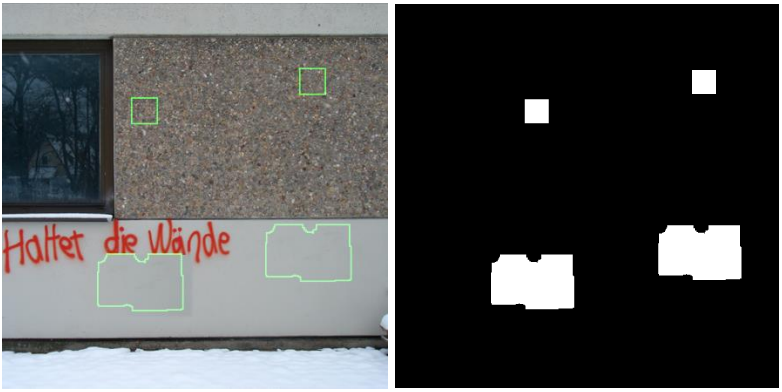
piksel dari metode *robust detection* sebagai fitur karakteristik dominan, dibantu dengan fitur *principal component* jika terdapat blok citra yang memiliki fitur perbandingan nilai piksel yang sama. Contoh keluaran dari algoritma *robust-duplication detection* dapat dilihat pada Gambar 5.66, Gambar 5.67, dan Gambar 5.68. Tiga citra hasil keluaran tersebut menunjukkan bahwa daerah yang terdeteksi lebih besar dan lebih halus dibanding kedua metode sebelumnya.

Tabel 5.20 *Confusion matrix* pada hasil uji coba 4 menggunakan metode *robust-duplication detection*

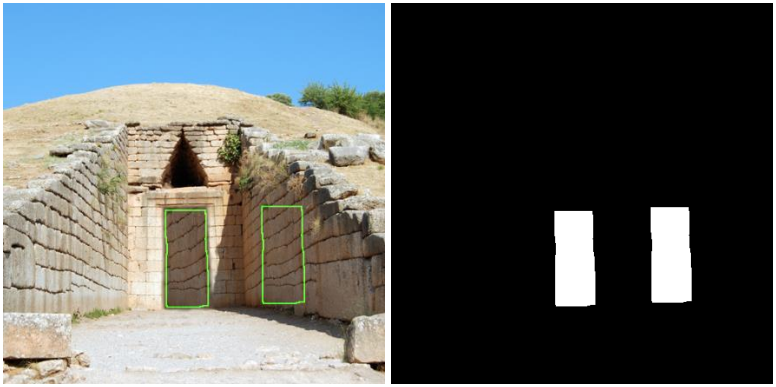
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_blur.png	1	0	0	0	100%
02_cattle_blur.png	1	0	0	0	100%
03_clean_walls_blur.copy	1	0	0	0	100%
04_fountain_blur.png	1	0	0	0	100%
05_four_babies_blur.png	1	0	0	0	100%
06_horses_blur.png	1	0	0	0	100%
07_knight_moves_blur.png	1	0	0	0	100%
08_lone_cat_blur.png	1	0	0	0	100%
09_malawi_blur.png	1	0	0	0	100%
10_mykene_blur.png	1	0	0	0	100%



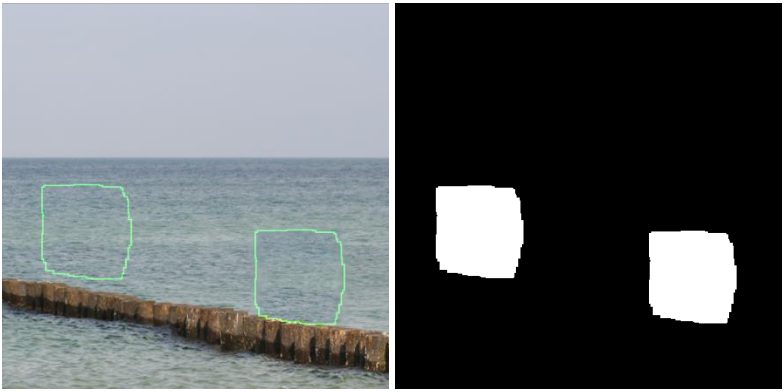
Gambar 5.60 Hasil deteksi dan hasil akhir citra masukan 1 dengan metode *duplication detection*



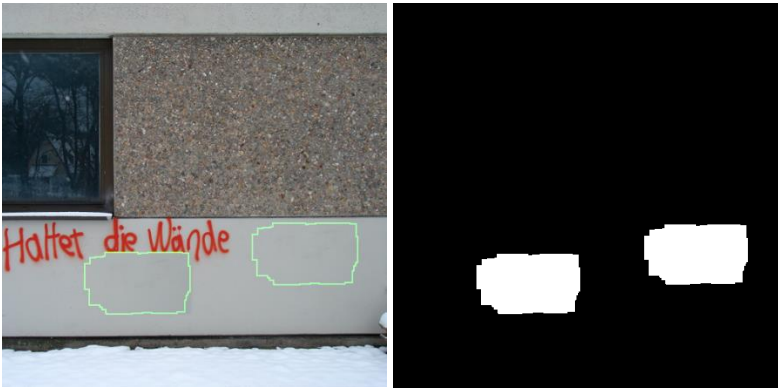
Gambar 5.61 Hasil deteksi dan hasil akhir citra masukan 3 dengan metode *duplication detection*



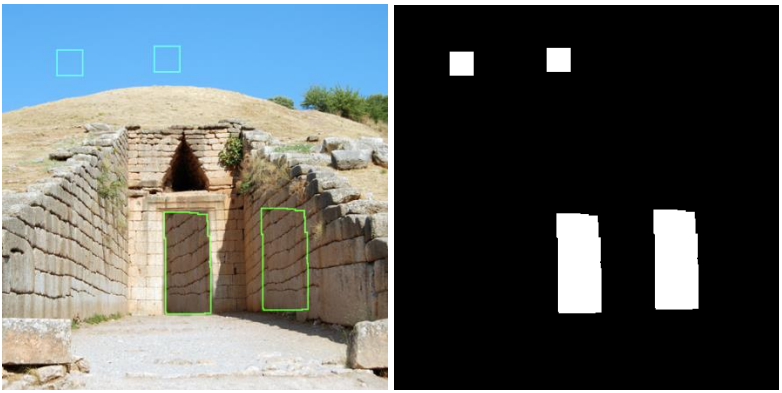
Gambar 5.62 Hasil deteksi dan hasil akhir citra masukan 10 dengan metode *duplication detection*



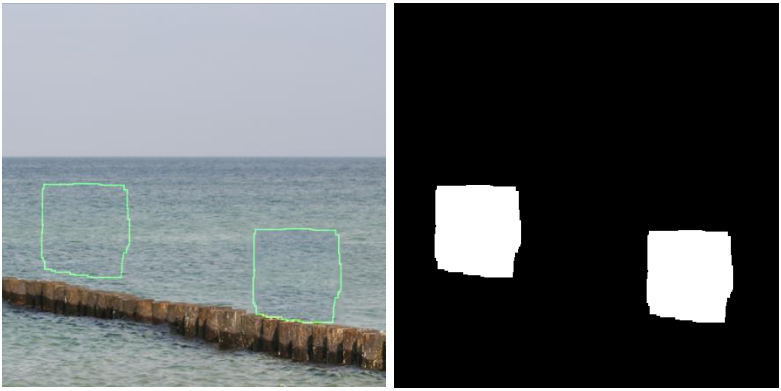
Gambar 5.63 Hasil deteksi dan hasil akhir citra masukan 1 dengan metode *robust detection*



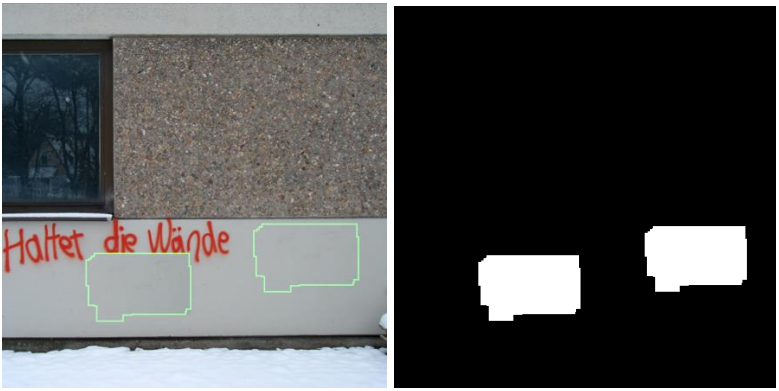
Gambar 5.64 Hasil deteksi dan hasil akhir citra masukan 3 dengan metode *robust detection*



Gambar 5.65 Hasil deteksi dan hasil akhir citra masukan 10 dengan metode *robust detection*



Gambar 5.66 Hasil deteksi dan hasil akhir citra masukan 1 dengan metode *robust-duplication detection*



Gambar 5.67 Hasil deteksi dan hasil akhir citra masukan 3 dengan metode *robust-duplication detection*

Gambar 5.68 Hasil deteksi dan hasil akhir citra masukan 10 dengan metode *robust-duplication detection*

5.3.5 Skenario Uji Coba 5

Skenario uji coba 5 adalah penghitungan akurasi pendeteksian *copy-move* pada citra dari *dataset* publik dengan menggunakan metode *duplication detection*, *robust detection*, dan modifikasi *duplication detection-robust detection*.

Nilai MSE, *similarity*, dan *confusion matrix* setiap metode pada hasil uji coba 5 terdapat pada Tabel 5.21, Tabel 5.22, Tabel 5.23, Tabel 5.24, dan Tabel 5.25. Berdasarkan nilai yang ditunjukkan pada beberapa tabel hasil, citra dari *dataset* publik akan efektif jika dilakukan pendeteksian dengan metode *robust detection* dengan rata-rata MSE sebesar **3772.86**, sementara metode *duplication detection* memiliki rata-rata MSE hingga **6881.08**.

Pada metode *duplication detection*, *principal component analysis* yang digunakan oleh algoritma ini dapat secara akurat memberikan fitur karakteristik blok citra, mengakibatkan halusnya tepi deteksi yang memang sesuai dengan tepian pada *ground truth*. Namun blok citra yang memiliki tekstur dan *offset* yang sama dengan salah satu daerah yang diduplikasi akan ikut masuk kedalam kelompok blok citra tersebut, mengakibatkan terjadinya

false positive yakni terduganya daerah yang sebenarnya bukan daerah yang terduplikasi. Hasil keluaran metode *duplication detection* terdapat pada Gambar 5.69, dan Gambar 5.70.

Tabel 5.21 Nilai MSE dari hasil uji coba 5 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_giraffe	9719.05	6935.39	6935.39
04_cattle	4043.12	610.32	608.90
Rata-rata	**6881.08**	**3772.86**	**3772.14**

Tabel 5.22 Nilai *smilarity* dari hasil uji coba 5 terhadap *ground truth*

Nama file	*duplication detection*	*robust detection*	*robust-duplication detection*
01_barrier	95.02	96.44	96.44
02_cattle	97.93	99.69	99.69
Rata-rata	**96.47%**	**98.07%**	**98.07%**

Pada metode *robust detection*, aturan perbandingan nilai piksel lebih handal dalam menangani persamaan tekstur antar blok citra, membuat tidak terjadinya *false positive* sama sekali. Namun diketahu bahwa aturan perbandingan nilai piksel memberikan fitur berupa perbandingan nilai piksel pada blok citra tersebut. Artinya, untuk blok citra yang terletak berdekatan maka fitur aturan perbandingan tersebut akan cenderung mirip. Hal ini mengakibatkan daerah yang terdeteksi akan sedikit meluas dan mengakibatkan daerah deteksi tidak halus. Hasil keluaran metode *robust detection* terdapat pada Gambar 5.71, dan Gambar 5.72.

Tabel 5.23 *Confusion matrix* pada hasil uji coba 5 dengan menggunakan metode *duplication detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_giraffe.png	3	1	2	0	50%
04_cattle.png	5	3	0	0	63%

Tabel 5.24 *Confusion matrix* pada hasil uji coba 5 dengan menggunakan metode *robust detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_giraffe.png	3	0	2	0	60%
04_cattle.png	5	0	0	0	100%

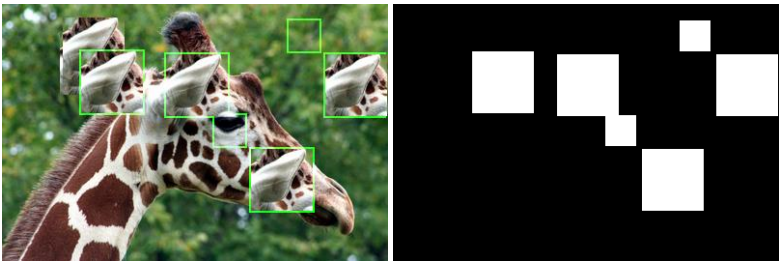
Tabel 5.25 *Confusion matrix* pada hasil uji coba 5 dengan menggunakan metode *robust-duplication detection*

Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_giraffe.png	3	0	2	0	60%
04_cattle.png	5	0	0	0	100%

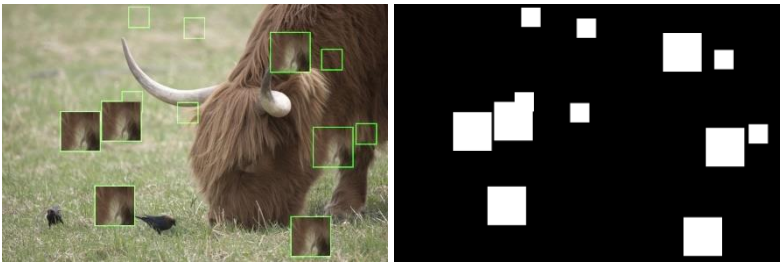
Secara umum, metode *robust detection* terlihat unggul dengan MSE yang lebih kecil, namun memiliki hasil tepian yang kasar. Disisi lain, hasil dari metode *duplication detection* sangat halus jika dilihat dari tepian hasil deteksi, namun metode ini sering melakukan kesalahan deteksi karena *principal component* pada blok citra yang berwarna homogen atau memiliki pola sama akan berdekatan, padahal blok citra tersebut tidak selalu hasil serangan *copy-move*. Tidak hanya *principal component* yang mirip, namun blok citra tersebut memiliki *offset* yang sama dengan blok citra lain yang memang hasil duplikasi dari serangan *copy-move*. Kesamaan pada *principal component* dan *offset* ini akan berakibat pada

masuknya pasangan blok citra tersebut kedalam kumpulan blok citra lain yang terduga hasil serangan *copy-move*.

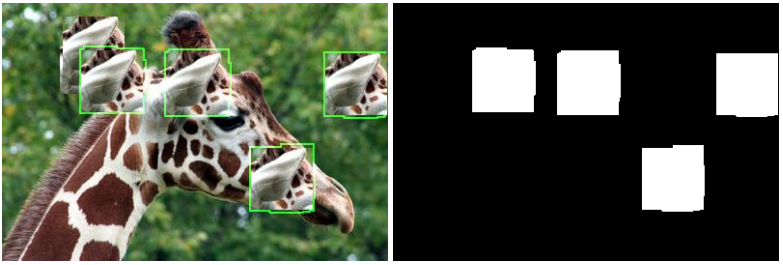
Algoritma gabungan kedua metode yakni *robust-duplication detection* memberikan toleransi kepada *duplication detection* yang diyakini lebih handal untuk dapat mendominasi proses deteksi pada citra *dataset* publik yang notabene sama dengan kasus serangan *copy-move* normal, tanpa menghilangkan fitur karakteristik dari metode *robust detection*. Hal ini mengakibatkan metode ini memiliki rata-rata MSE yang mendekati *duplication detection* yakni sebesar **3772.14**. Metode ini menghasilkan dugaan daerah duplikasi yang lebih halus dibandingkan hasil dari metode *robust detection* karena menggunakan *principal component* dari metode *duplication detection* sebagai salah satu fitur karakteristiknya. Disisi lain, metode ini juga tidak melakukan *false positive* karena menggunakan fitur karakteristik berupa aturan perbandingan nilai piksel dari metode *robust detection*. Hasil keluaran metode *robust-duplication detection* terdapat pada Gambar 5.73, dan Gambar 5.74.



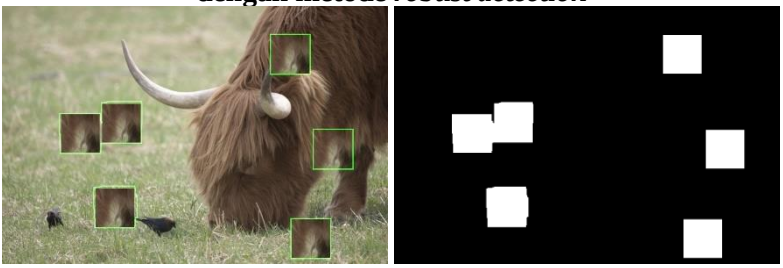
Gambar 5.69 Hasil deteksi dan hasil akhir citra masukan 1 dengan metode *duplication detection*



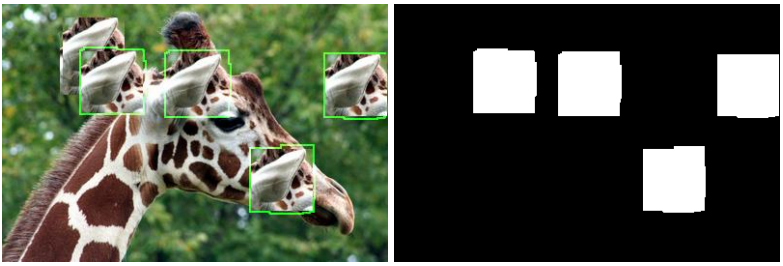
Gambar 5.70 Hasil deteksi dan hasil akhir citra masukan 2 dengan metode *duplication detection*



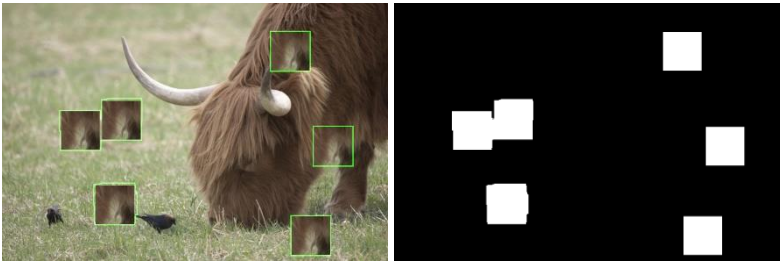
Gambar 5.71 Hasil deteksi dan hasil akhir citra masukan 1 dengan metode *robust detection*



Gambar 5.72 Hasil deteksi dan hasil akhir citra masukan 2 dengan metode *robust detection*



Gambar 5.73 Hasil deteksi dan hasil akhir citra masukan 1 dengan metode *robust-duplication detection*



Gambar 5.74 Hasil deteksi dan hasil akhir citra masukan 2 dengan metode *robust-duplication detection*

5.3.6 Skenario Uji Coba 6

Skenario uji coba 6 adalah penghitungan parameter loncatan *overlapping block* terbaik pada metode *robust-duplication detection*. Parameter loncatan *overlapping block* digunakan untuk menentukan seberapa banyak loncatan yang terjadi pada saat pembuatan *overlapping block*. Parameter ini memiliki nilai bawaan satu, dengan arti dilakukan perulangan untuk pembuatan *overlapping block* dengan *increment* satu satuan. Pemberian nilai selain satu satuan pada parameter ini mengakibatkan metode akan melangkahi sebagian *overlapping block*, menjadikan berkurangnya jumlah *overlapping block* yang didapatkan, sehingga waktu jalannya metode akan semakin cepat.

Skenario dilakukan dengan memasukkan sepuluh citra *dataset* pada Subbab 5.2.2 dan menjalankan metode *robust-duplication detection* dengan parameter loncatan *overlapping block* sebesar satu, dua, dan tiga satuan. Hasil keluaran metode dengan parameter loncatan *overlapping block* sebesar satu satuan terdapat pada Gambar 5.75 dan Gambar 5.76. Hasil keluaran metode dengan parameter loncatan *overlapping block* sebesar dua satuan terdapat pada Gambar 5.77 dan Gambar 5.78. Hasil keluaran metode dengan parameter loncatan *overlapping block* sebesar tiga satuan terdapat pada Gambar 5.79 dan Gambar 5.80.

Rataan MSE citra keluaran berdasarkan nilai parameter loncatan *overlapping block* terdapat pada Tabel 5.26. Dari hasil pada tabel tersebut, didapatkan bahwa semakin besar nilai parameter loncatan *overlapping block*, semakin besar pula MSE yang dihasilkan. Hal ini dapat terjadi karena loncatan pembuatan *overlapping block* akan membuat hasil *overlapping block* menjadi lebih renggang antar satu dengan yang lainnya. Meningkatnya kerenggangan antar blok ini mengakibatkan berkurangnya akurasi, karena terdapat daerah citra tertentu yang sebenarnya merupakan daerah *copy-move* namun tidak terdeteksi karena terlompati saat pembuatan *overlapping block*. Nilai MSE terbaik didapatkan dengan penggunaan nilai satu satuan pada parameter loncatan *overlapping block*. Nilai satu satuan ini akan memaksimalkan jumlah *overlapping block* yang terbuat sehingga kumpulan *overlapping block* dapat menjangkau setiap piksel pada citra. Meningkatnya jangkauan dari *overlapping block* membuat semua daerah citra *copy-move* dapat ikut terdeteksi dengan baik.

Rataan *similarity* citra keluaran berdasarkan nilai parameter loncatan *overlapping block* terdapat pada Tabel 5.27. Berdasarkan isi tabel tersebut, didapatkan bahwa semakin besar nilai parameter loncatan *overlapping block* maka semakin kecil *similarity* antara citra yang dihasilkan dengan citra *ground truth*. Mirip seperti pada penjelasan sebelumnya, menurunnya nilai *similarity* ini terjadi karena adanya loncatan pembuatan *overlapping block*. Loncatan pembuatan yang terlalu besar akan

membuat hasil *overlapping block* menjadi lebih renggang antar satu dengan yang lainnya. Meningkatnya kerenggangan antar blok ini sama artinya dengan menurunnya jangkauan *overlapping block* pada citra masukan. Penurunan jangkauan ini akan menurunkan ketajaman citra hasil, dikarenakan citra hasil dibuat berdasarkan koordinat *overlapping block* yang terbuat. Meski metode dapat mendeteksi sebagian daerah *copy-move*, namun konstruksi citra hasil tidak akan setajam jika menggunakan seluruh *overlapping block* yang dapat dihasilkan.

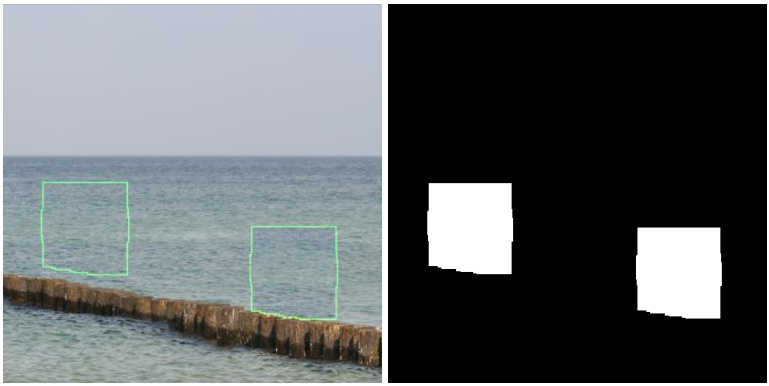
Tabel 5.26 Nilai MSE dari hasil uji coba 6 terhadap *ground truth*

Nama file	Banyak lompatan		
	1 blok	2 blok	3 blok
01_barrier_copy.png	250.04	791.78	824.52
02_cattle_copy.png	410.77	906.38	19682.82
03_clean_walls_copy.png	80.37	16274.61	16274.61
04_fountain_copy.png	190.50	16399.62	16399.62
05_four_babies_copy.png	235.15	352.73	471.79
06_horses_copy.png	117.58	15476.87	15476.87
07_knight_moves_copy.png	1704.11	11884.11	2082.14
08_lone_cat_copy.png	870.66	4406.87	21285.73
09_malawi_copy_png	1935.01	4687.70	3774.86
10_mykene_copy.png	5.95	12661.00	12661.00
Rataan MSE	**580.01**	**8384.17**	**10893.40**

Rataan waktu yang diperlukan untuk menghasilkan citra keluaran berdasarkan nilai parameter loncatan *overlapping block* terdapat pada Tabel 5.28. Berdasarkan isi tabel tersebut, didapatkan bahwa semakin besar nilai parameter loncatan *overlapping block*, maka semakin cepat waktu yang dibutuhkan metode untuk mendeteksi citra. Hal ini dapat terjadi karena loncatan pada pembuatan *overlapping block* akan mengurangi jumlah *overlapping block* yang dihasilkan pada sebuah citra.

Berkurangnya jumlah *overlapping block* akan berakibat pada berkurangnya waktu yang dibutuhkan untuk mencari fitur karakteristik pada setiap *overlapping block*, juga pada pengolahan *overlapping block* saat masuk ke tahap analisa.

Confusion matrix beserta akurasi metode berdasarkan nilai loncatan pembuatan *overlapping block* yang berbeda terdapat pada Tabel 5.29, Tabel 5.30, dan Tabel 5.31. Saat metode melakukan loncatan satu blok pada pembuatan *overlapping block*, didapatkan rataan akurasi sebesar 100% dengan hasil *true positive* pada semua citra dan tidak terjadi *false negative*. Pada saat dilakukan loncatan sebesar dua satuan, rataan akurasi turun menjadi 43% disertai dengan turunnya *true positive* dan meningkatnya *false negative*. Pada saat dilakukan loncatan sebesar tiga satuan, rataan akurasi turun lagi menjadi 35% dengan makin berkurangnya *true positive*. Turunnya akurasi ini dapat terjadi akibat kerenggangan *overlapping block* yang terbuat, karena kerenggangan tersebut akan menurunkan jangkauan *overlapping block* pada daerah citra. Penurunan jangkauan *overlapping block* ini tidak hanya menurunkan ketajaman citra hasil namun juga menurunkan akurasi deteksi karena akan ada bagian citra yang terlewati dan metode tidak dapat medeteksi blok terlewat sebagai daerah *copy-move*.



Gambar 5.75 Hasil deteksi dan hasil akhir citra masukan 1 dengan lompatan satu satuan

Tabel 5.27 Nilai *smilarity* dari hasil uji coba 6 terhadap *ground truth*

Nama file	Banyak lompatan		
	1 blok	2 blok	3 blok
01_barrier_copy.png	99.87%	99.59%	99.58%
02_cattle_copy.png	99.79%	99.54%	89.91%
03_clean_walls_copy.png	99.96%	91.66%	91.66%
04_fountain_copy.png	99.90%	91.59%	91.59%
05_four_babies_copy.png	99.88%	99.82%	99.76%
06_horses_copy.png	99.94%	92.07%	92.07%
07_knight_moves_copy.png	99.13%	93.91%	98.93%
08_lone_cat_copy.png	99.55%	97.74%	89.09%
09_malawi_copy_png	97.85%	96.43%	96.90%
10_mykene_copy.png	100.00%	93.51%	93.51%
Rataan *similarity*	**99.59%**	**95.59%**	**94.30%**

Tabel 5.28 Waktu metode untuk mendeteksi citra masukan

Nama file	Banyak lompatan		
	1 blok	2 blok	3 blok
01_barrier_copy.png	0:27:51	0:01:55	0:00:49
02_cattle_copy.png	0:26:03	0:01:58	0:00:46
03_clean_walls_copy.png	0:27:14	0:02:00	0:00:46
04_fountain_copy.png	0:26:39	0:02:00	0:00:45
05_four_babies_copy.png	0:26:49	0:01:53	0:00:46
06_horses_copy.png	0:26:41	0:01:52	0:00:46
07_knight_moves_copy.png	0:26:58	0:01:56	0:00:47
08_lone_cat_copy.png	0:26:47	0:01:52	0:00:46
09_malawi_copy_png	0:26:50	0:01:57	0:00:46
10_mykene_copy.png	0:26:42	0:01:56	0:00:45
Rataan waktu	**0:26:51**	**0:01:56**	**0:00:46**

Tabel 5.29 *Confusion matrix* pada hasil uji coba 6 dengan lompatan satu blok

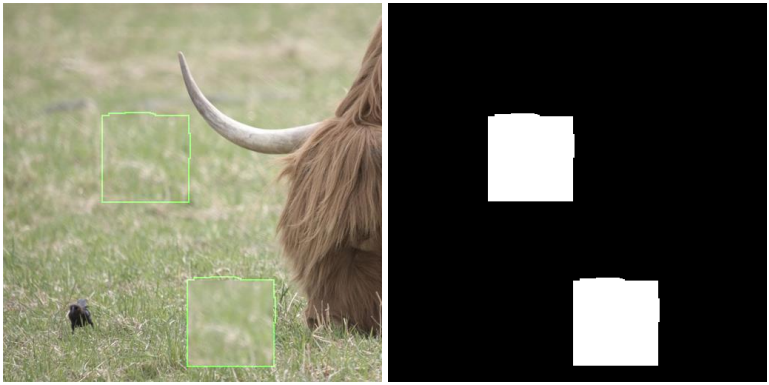
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_copy.png	1	0	0	0	100%
02_cattle_copy.png	1	0	0	0	100%
03_clean_walls_copy.png	1	0	0	0	100%
04_fountain_copy.png	1	0	0	0	100%
05_four_babies_copy.png	1	0	0	0	100%
06_horses_copy.png	1	0	0	0	100%
07_knight_moves_copy.png	1	0	0	0	100%
08_lone_cat_copy.png	1	0	0	0	100%
09_malawi_copy_png	1	0	0	0	100%
10_mykene_copy.png	1	0	0	0	100%
Rataan akurasi					**100%**

Tabel 5.30 *Confusion matrix* pada hasil uji coba 6 dengan lompatan dua blok

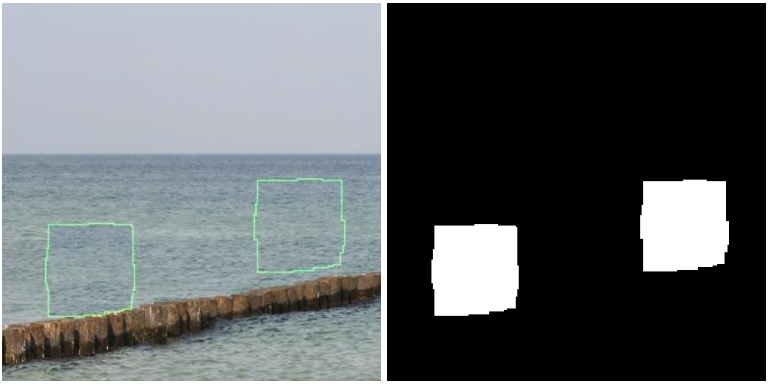
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_copy.png	1	0	0	0	100%
02_cattle_copy.png	1	0	0	0	100%
03_clean_walls_copy.png	0	0	1	0	0%
04_fountain_copy.png	0	0	1	0	0%
05_four_babies_copy.png	1	0	0	0	100%
06_horses_copy.png	0	0	1	0	0%
07_knight_moves_copy.png	1	0	1	0	50%
08_lone_cat_copy.png	1	0	2	0	33%
09_malawi_copy_png	1	0	1	0	50%
10_mykene_copy.png	0	0	1	0	0%
Rataan akurasi					**43%**

Tabel 5.31 *Confusion matrix* pada hasil uji coba 6 dengan lompatan tiga blok

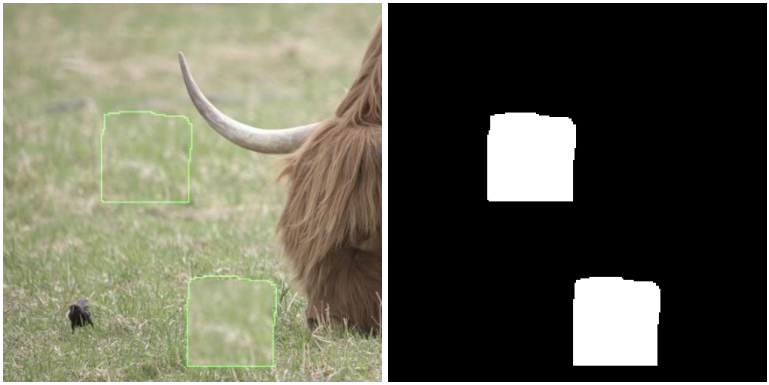
Nama file	*Confusion matrix*				Akurasi
	TP	FP	FN	TN	
01_barrier_copy.png	1	0	0	0	100%
02_cattle_copy.png	0	0	1	0	0%
03_clean_walls_copy.png	0	0	1	0	0%
04_fountain_copy.png	0	0	1	0	0%
05_four_babies_copy.png	1	0	0	0	100%
06_horses_copy.png	0	0	1	0	0%
07_knight_moves_copy.png	1	0	0	0	100%
08_lone_cat_copy.png	0	0	1	0	0%
09_malawi_copy_png	1	0	1	0	50%
10_mykene_copy.png	0	0	1	0	0%
Rataan akurasi					**35%**



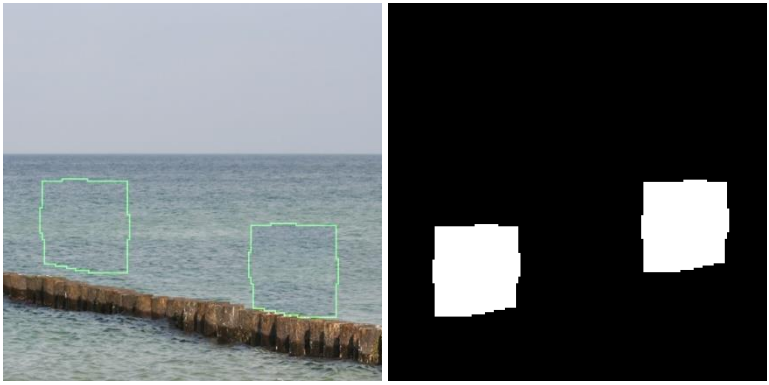
Gambar 5.76 Hasil deteksi dan hasil akhir citra masukan 2 dengan lompatan satu satuan



Gambar 5.77 Hasil deteksi dan hasil akhir citra masukan 1 dengan lompatan dua satuan



Gambar 5.78 Hasil deteksi dan hasil akhir citra masukan 2 dengan lompatan dua satuan



Gambar 5.79 Hasil deteksi dan hasil akhir citra masukan 1 dengan lompatan tiga satuan

Gambar 5.80 Hasil deteksi dan hasil akhir citra masukan 2 dengan lompatan tiga satuan

5.4 Evaluasi Umum Skenario Uji Coba

Pertama, skenario uji coba 1 digunakan untuk menentukan parameter optimal pada metode yang digunakan. Dari skenario uji coba 1, didapatkan bahwa nilai parameter yang paling optimal adalah *Nn* sebesar 2, *Nd* sebesar 50 dan *Nf* sebesar 750. Kombinasi

nilai parameter tersebut dikatakan optimal berdasarkan hasil terbaik yang didapat dari melakukan percobaan dengan berbagai nilai parameter. Nilai parameter optimal akan berganti seiring dengan ukuran citra yang digunakan. Kombinasi nilai parameter ini kemudian digunakan untuk mendapatkan hasil pada skenario uji coba selanjutnya.

Pembahasan skenario uji coba dua hingga lima akan merujuk pada Tabel 5.32. Berdasarkan skenario uji coba dua yang telah dilakukan, dapat diketahui bahwa tingkat ketepatan ketiga metode yakni *duplication detection*, *robust detection*, dan *robust-duplication detection* dalam mendeteksi *copy-move* pada citra otentik sudah sangat baik dengan tingkat akurasi mencapai **100%** pada masing-masing metode. Kesimpulan yang didapat adalah ketiga metode tersebut mampu mengenali citra yang memang otentik, meskipun pada citra tersebut terdapat banyak dareah dengan warna homogen yang sekilas terlihat sama.

Berdasarkan skenario uji coba tiga yang telah dilakukan, dapat diketahui bahwa tingkat ketepatan ketiga metode dalam mendeteksi *copy-move* pada citra biasa dimenangkan oleh metode *duplication detection* dengan rataan MSE sebesar **542.40**, rataan *similarity* sebesar 99.60%, dan rataan akurasi sebesar 83%. Dilanjutkan metode *robust-duplication detection* dengan rataan MSE sebesar **580.01**, rataan *similarity* sebesar 99.59% dan rataan akurasi sebesar 100%. Terakhir oleh metode *robust detection* dengan rataan MSE sebesar **1023.98**, rataan *similarity* sebesar 99.37%, dan rataan akurasi sebesar 90%. Pada kasus citra masukan yang tidak dilakukan *blurring*, terbukti metode *robust-duplication detection* dapat beradaptasi dengan menghasilkan jawaban yang mendekati metode *duplication detection.*

Berdasarkan skenario uji coba empat yang telah dilakukan, dapat diketahui bahwa tingkat ketepatan ketiga metode dalam mendeteksi *copy-move* pada citra yang dilakukan *blurring* dimenangkan oleh metode *robust-duplication detection* dengan rataan MSE sebesar **2002.91**, rataan *similarity* sebesar 98.86%, dan rataan akurasi sebesar 100%. Dilanjutkan metode *robust detection*

dengan rataan MSE sebesar **2538.93**, rataan *similarity* sebesar 98.60%, dan rataan akurasi sebesar 80%. Terakhir oleh metode *duplication detection* dengan rataan MSE sebesar **7504.83**, rataan *similarity* sebesar 96.03%, dan rataan akurasi sebesar 75%. Pada kasus citra masukan yang dilakukan *blurring*, terbukti metode *robust-duplication detection* dapat beradaptasi dengan menghasilkan jawaban yang mendekati hasil metode *robust detection*. Bahkan modifikasi kedua metode tersebut menghasilkan jawaban yang lebih baik dari masing-masing metode awal.

Berdasarkan ketiga skenario uji coba diatas, disimpulkan bahwa metode *robust-duplication detection* dapat beradaptasi sesuai dengan operasi yang telah dilakukan pada citra masukan, menjadikannya handal dalam mendeteksi *copy-move* baik pada serangan biasa maupun serangan pada tingkat yang lebih tinggi yang melibatkan *post region duplication process*.

Untuk skenario lima, dapat diketahui bahwa sebenarnya *dataset* yang digunakan memiliki kondisi sama seperti skenario uji coba dua, yakni citra yang diserang *copy-move* biasa. Metode *robust-duplication detection* menjadi metode yang paling efektif pada pendeteksian serangan *copy-move* dengan rataan MSE sebesar **3772.14**, rataan *similarity* sebesar 98.07%, dan rataan akurasi sebesar 80%. Dilanjutkan metode *robust detection* dengan rataan MSE sebesar **3772.86**, rataan *similarity* sebesar 98.07%, dan rataan akurasi sebesar 80%. Terakhir adalah metode *duplication detection* dengan rataan MSE sebesar **6881.08**, rataan *similarity* sebesar 96.47%, dan rataan akurasi sebesar 56%. Pada kasus citra masukan yang tidak dilakukan *blurring*, metode akan menggunakan fitur dari metode *duplication detection* untuk memperhalus tepian, dan menggunakan fitur dari *robust detection* untuk mengurangi *false positive* yang terjadi.

Pada skenario uji coba enam, semakin banyak loncatan yang terjadi pada pembuatan *overlapping block*, maka waktu yang dibutuhkan metode untuk melakukan deteksi akan semakin berkurang. Akurasi akan turun dengan berkurangnya *true positive* dan meningkatnya *false negative*. Turunnya akurasi berakibat pada

naiknya MSE dan berkruangnya *similarity*. Tabel perbandingan hasil metode pada skenario ini terdapat pada Tabel 5.33.

Tabel 5.32 Perbandingan hasil metode berdasarkan skenario uji coba

Skenario		**metode**		
		duplication detection	*robust detection*	*robust-duplication detection*
citra otentik	Akurasi	100%	100%	100%
	MSE	0	0	0
	Similarity	100%	100%	100%
citra *copy-move*	Akurasi	83%	90%	100%
	MSE	542.40	1023.98	580.01
	Similarity	99.60%	99.37%	99.59%
citra *copy-move* + *blur*	Akurasi	75%	80%	100%
	MSE	7504.83	2538.93	2002.91
	Similarity	96.03%	98.60%	98.86%
citra *dataset* publik	Akurasi	56%	80%	80%
	MSE	6881.08	3772.86	3772.14
	Similarity	96.47%	98.07%	98.07%

Tabel 5.33 Perbandingan hasil metode berdasarkan parameter loncatan pada pembuatan *overlapping block*

Jumlah loncatan	**Hasil**		
	Akurasi	MSE	*Similarity*
1 blok	96.67%	580.01	99.59%
2 blok	43.33%	8384.17	95.59%
3 blok	35.00%	10893.40	94.30%

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 metode.

6.1 Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil uji coba pendeteksian *copy-move* pada berbagai variasi citra menggunakan metode *duplication detection*, *robust detection*, dan *robust-duplication detection* adalah sebagai berikut:

1. Implementasi metode *duplication detection* dapat digunakan sebagai salah satu metode untuk mendeteksi dan menganalisis serangan *copy-move* pada citra yang tidak dilakukan *post region duplication process*. Modifikasi metode tersebut berupa penggunaan nilai *offset* yang tidak dilakukan operasi *absolut* dapat menambah akurasi.
2. Implementasi metode *robust detection* dapat digunakan sebagai salah satu metode untuk mendeteksi dan menganalisis serangan *copy-move* pada citra yang dilakukan *post region duplication process*. Modifikasi metode tersebut berupa penggantian penggunaan *histogram* menjadi penggunaan *threshold* minimal dapat meningkatkan kemampuan metode dalam mendeteksi serangan *copy-move* yang dilakukan pada lebih dari satu pasang daerah.
3. Implementasi metode *robust-duplication detection* dapat digunakan sebagai salah satu metode untuk mendeteksi dan menganalisis serangan *copy-move* baik pada pada citra biasa maupun citra yang dilakukan *post region duplication process*. Metode tersebut dapat beradaptasi sesuai dengan citra masukan. Saat citra masukan bebas dari operasi *blurring*, maka metode akan memenangkan *principal component* sebagai fitur dominan. Sementara jika citra

masukan dilakukan proses *blurring*, maka aturan perbandingan nilai piksel akan digunakan sebagai fitur dominan. Modifikasi metode *duplication detection* dan *robust detection* pada metode ini serta penambahan teknik toleransi dengan mengimplementasikan fungsi pembulatan (*round*) pada fitur karakteristik terbukti meningkatkan kehandalan metode dalam mendeteksi serangan *copy-move* pada berbagai variasi citra masukan.

4. Pada kasus tugas akhir ini, akurasi terbaik dalam pendeteksian citra yang bebas dari *post region duplication process* didapat dengan pendeteksian menggunakan metode *duplication detection* dan metode *robust-duplication detection* yang memiliki hasil bermiripan. Sedangkan akurasi terbaik dalam pendeteksian citra yang melibatkan *post region duplication process* didapat dengan pendeteksian menggunakan metode *robust-duplication detection.*
5. Akurasi tertinggi yang didapat dari seluruh skenario uji coba adalah 100% dengan *similarity* sebesar 99.55% yang berasal dari penggunaan metode *robust-duplication detection*.
6. Ketiga metode yang digunakan memiliki rataan MSE sebesar 2384.93, rataan *similarity* sebesar 98.7%, dan rataan akurasi sebesar 87%.
7. Ketiga metode yang digunakan yakni *duplication detection*, *robust detection*, dan *robust-duplication detection* tidak dapat mendeteksi daerah serangan *copy-move* yang saling bertindih.
8. Semakin besar loncatan yang terjadi pada pembuatan *overlapping block*, maka semakin berkurang waktu yang dibutuhkan metode untuk melakukan proses deteksi pada citra, dengan *trade-off* akurasi yang menurun.

6.2 Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah dapat dicoba penggabungan metode pemberian toleransi yang lain seperti *weighting factor* untuk lebih meningkatkan toleransi antar kedua metode yang digunakan.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] K. K. Light. Fonda dan *P*. Fakery, *The Washington Post, Saturday, P*. A21, 28 February 2004.

[2] "Sorry.. we were hoaxed: Iraqi PoW abuse pictures handed to us WERE fake," *Daily Mirror Newspaper*, 15 May 2004.

[3] W. Luo dan J. Huang, "Robust Detection of Region-Duplication Forgery in Digital Image," *IEEE The 18th International Confrence on Pattern Recognition (ICPR'06),* 2006.

[4] C.-Y. Lin dan S.-F. Chang, "A Robust Image Authentication Method Distinguishing JPEG Compression from Malicious Manipulation," *IEEE Transactions On Circuits and Systems of Video Technology,* vol. 11, pp. 153-168, 2001.

[5] I. The MathWorks, "Add noise to image - MATLAB imnoise," The MathWorks, Inc, [Online]. Available: http://www.mathworks.com/help/images/ref/imnoise.html ?requestedDomain=www.mathworks.com. [Diakses 28 5 2017].

[6] H. Farid, A pictures tells a thousands lies, New Scientist, 6 September 2003.

[7] A. C. Popescu dan H. Farid, "Exposing Digital Forgeries by Detecting Duplicated Image Regions," Darmouth College, Hanover, 2004.

[8] A. Popescu dan H. Farid, "Exposing Digital Forgeries in Color Filter *Array* Interpolated Images," *IEEE Trans. on signal processing,* pp. 3948-3959, 2005.

[9] A. Popescu dan H. Farid, "Exposing Digital Forgeries by Detecting Traces of Resampling," *IEEE Trans. on signal processing,* pp. 758-767, 2005.

[10] J. Fridrich, D. Soukal dan J. Lukas, "Detection of Copy-Move Forgery in digitals Images," *Proc. of Digital Forensic Research Workshop,* 2003.

[11] T.-T. Ng dan S.-F. Chang, dalam *A Model for Image Splicing,* ICIP, pp. 1169-1172 Vol.2.

[12] B. D. Carrier, "Basic Digital Forensic Investigation Concepts," Brian Carrier , 7 June 2006. [Online]. Available: http://www.digital-evidence.org/di_basics.html. [Diakses 27 October 2017].

[13] 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.

[14] H. Omrani dan R. G. Beiragh, "Performance Assessment of Iranian Electricity Distribution Companies by an Integrated Cooperative Game Data Envelopment Analysis Principal Component Analysis Approach," *International Journal of Electrical Power and Energy Systems,* vol. 64, pp. 617-625, 2015.

[15] D. Vogan, "Lexicographic Order," Massachusetts Institute of Technology, [Online]. Available: http://www-math.mit.edu/~dav/lex2.pdf. [Diakses 29 5 2017].

[16] "Anaconda | Continuum Analytics: Documentation," Anaconda, [Online]. Available: https://docs.continuum.io/anaconda/index. [Diakses 16 June 2017].

[17] "OpenCV," OPEN SOURCE COMPUTER VISION, [Online]. Available: http://opencv.org/. [Diakses 16 June 2017].

[18] S. Community, "NumPy v1.10 Manual," 18 October 2015. [Online]. Available: http://docs.scipy.org/doc/numpy-1.10.1/index.html. [Diakses May 2017].

[19] "SciPy," Scientific Computing Tools for Python, [Online]. Available: https://scipy.org/. [Diakses 16 June 2017].

[20] “Scikit-Learn,” Scikit-Learn, [Online]. Available: http://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error. [Diakses 16 June 2017].

[21] C. Riess dan J. Jordan, “Image Manipulation Dataset,” University of Erlangen-Nuremberg, [Online]. Available: https://www5.cs.fau.de/research/data/image-manipulation/. [Diakses 30 5 2017].

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Rahmat Nazali Salimi lahir di Surakarta pada tanggal 19 Februari 1995. Penulis menempuh pendidikan formal dimulai dari TK Baiturrahman Sukoharjo (2000-2001), SDIT Nur Hidayah Surakarta (2001-2007), SMPIT Nur Hidayah Surakarta (2007-2010), SMAN 3 Surakarta (2010-2013) dan S1 Teknik Informatika ITS (2012-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis memiliki minat pada pengolahan citra digital terutama aplikasinya untuk melakukan deteksi pemalsuan citra. Penulis suka menikmati waktu luangnya untuk berkontribusi pada kode milik orang lain pada situs GitHub. Penulis dapat dihubungi melalui surel pribadi pada rahmatnazali95@gmail.com atau melalui surel formal pada rahmat13@mhs.if.its.ac.id.