Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

FORENSIK DIGITAL DETEKSI PEMALSUAN *COPY-MOVE* CITRA DENGAN MENGGUNAKAN METODE *BLOCK MATCHING*

SAID AL MUSAYYAB
NRP 5112100177

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 2016

TUGAS AKHIR – KI141502

FORENSIK DIGITAL DETEKSI PEMALSUAN *COPY-MOVE* CITRA DENGAN MENGGUNAKAN METODE *BLOCK MATCHING*

SAID AL MUSAYYAB
NRP 5112100177

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 2016

UNDERGRADUATE THESIS – KI141502

Digital Forensic of Copy-Move Image Forgery Detection using Block Matching Method

SAID AL MUSAYYAB
NRP 5112100177

Supervisor I
Tohari Ahmad, S.Kom., MIT., Ph.D.

Supervisor II
Hudan Studiawan, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2016

LEMBAR PENGESAHAN

FORENSIK DIGITAL DETEKSI PEMALSUAN *COPY-MOVE* CITRA DENGAN MENGGUNAKAN METODE *BLOCK MATCHING*

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
SAID AL MUSAYYAB
NRP: 5112 100 177

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

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

SURABAYA
Juni, 2016

FORENSIK DIGITAL DETEKSI PEMALSUAN *COPY-MOVE* CITRA DENGAN MENGGUNAKAN METODE BLOCK MATCHING

Nama Mahasiswa	**: SAID AL MUSAYYAB**
NRP	**: 5112100177**
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

Semakin berkembangnya teknologi manipulasi citra, semakin mudah seseorang untuk melakukan pemalsuan citra. Pemalsuan citra dapat menimbulkan kesalahpahaman bagi seseorang yang melihat citra tersebut. Salah satu jenis pemalsuan citra yaitu copy-move, di mana bagian pada citra disalin, kemudian ditempelkan pada bagian lain pada citra yang sama. Tugas akhir ini akan mengimplementasikan deteksi pemalsuan copy-move dengan menggunakan metode block matching. Metode block matching dibagi menjadi dua pendekatan, yaitu exact match dan robust match. Kedua metode akan mengambil blok ke seluruh bagian citra, kemudian membandingkan blok-blok yang ditemui. Exact match mengambil blok melalui representasi piksel, sedangkan robust match mengambil blok melalui nilai DCT (Discrete cosine transform). Blok yang sama akan dideteksi sebagai serangan copy-move. Metode robust match dapat menghasilkan kualitas yang lebih baik dibandingkan dengan exact match. Dari sampel data set yang digunakan, robust match menghasilkan kualitas rata-rata 75% dengan hasil terbaik sebesar 97%.

Kata kunci: Forensik Digital, Copy-move, Citra

Digital Forensic of *Copy-Move* Image Forgery Detection using Block Matching Method

Student's Name	**: SAID AL MUSAYYAB**
Student's ID	**: 5112100177**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Tohari Ahmad S.Kom., MIT., Ph.D**
Second Advisor	**: Hudan Studiawan, S.Kom., M.Kom.**

Abstract

In recent years, development of image editing makes people easier to manipulate image. Image manipulation can raise misleading to someone looking at the image. One example of image forgery is copy-move, where some region of image copied and pasted on other position in the same image. This final project will implement copy-move forgery detection using block matching method. Block matching method devided into two approaches, exact match and robust match. Block matching method will generate overlapping blocks to all regions of image. Then it will compare the blocks, and identic blocks will detected as copy-move area. Exact match will use pixel representation and robust match will use DCT (Discrite Cosine Transform) for generating blocks. From the experiments, robust match produce average quality 75% with the best up to 95% which is better than exact match.

Keywords: Digital Forensic, Copy-Move, Image

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **"Forensik Digital Deteksi Pemalsuan *Copy-Move* Citra dengan Menggunakan Metode *Block Matching*".** Selama pengerjaan Tugas Akhir, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menjalani perkuliahan di Teknik Informatika ITS.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga tercinta yang selalu memberikan dukungan penuh agar dapat terselesainya Tugas Akhir ini.
3. Bapak Tohari Ahmad S.Kom., MIT., Ph.D. selaku pembimbing I yang telah membimbing dan membantu penulis dalam menyelesaikan Tugas Akhir.
4. Bapak Hudan Studiawan, S.Kom., M.Kom. selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi penulis dalam mengerjakan Tugas Akhir ini.
5. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga, penulis mengharapkan kritik dan saran yang membangun dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2016

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada era digital ini, banyak hal yang dapat disimpan ke dalam penyimpanan digital seperti dokumen, foto, video, dan hal lainnya. Penyimpanan media digital sangatlah penting, karena dengan digitalisasi, hal-hal yang disimpan tersebut dapat disimpan dalam jangka waktu yang lama tanpa takut rusak, mudah untuk diduplikasi, dan juga mudah untuk dimodifikasi. Dengan mudahnya duplikasi dan modifikasi dokumen digital, terdapat banyaknya penyalahgunaan terhadap dokumen tersebut seperti pembajakan, pemalsuan dokumen, pemalsuan citra, dan lain-lain.

Perkembangan teknologi mengenai pemrosesan citra semakin mempermudah pengguna untuk melakukan modifikasi dan memalsukan citra. Dampak dari modifikasi citra dapat terciptanya situasi baru yang berbeda dengan citra asli. Hal ini dapat menimbulkan kesalahpahaman bagi seseorang yang melihat citra tersebut. Orang yang melihat citra tersebut juga merasa kesulitan untuk membedakan apakah citra tersebut asli atau adanya modifikasi. Oleh karena itu dibutuhkannya suatu metode yang dapat mendeteksi keaslian suatu citra digital.

Terdapat banyak macam pemalsuan citra, salah satu di antaranya adalah *copy-move*. *Copy-move* adalah cara di mana pengguna menyalin suatu bagian citra lalu menempelkannya di bagian lain pada citra yang sama [1]. Teknik ini biasanya digunakan untuk menutupi objek yang dianggap pengganggu dengan cara menempelkan objek yang serupa di sekitarnya. Selain itu *copy-move* juga digunakan untuk menduplikasi suatu objek pada citra. Sehingga terdapat objek serupa yang lebih banyak dari citra asli. Penelitian ini akan mendeteksi serangan *copy-move* dengan menggunakan metode *block matching*. *Block matching* berjalan dengan cara membagi citra menjadi

beberapa blok lalu dilakukan operasi perhitungan dan perbandingan antar blok. Metode *block matching* akan dilakukan melalui dua pendekatan, yaitu *exact match* dan *robust match*. *Exact match* akan mendeteksi objek berdasarkan representasi blok piksel pada citra. Sedangkan *robust match* akan mendeteksi objek berdasarkan kuantisasi DCT (*Discrete cosine transform*) pada blok citra. Kedua pendekatan ini akan dibandingkan untuk diketahui mana yang lebih baik dalam mendeteksi serangan *copy-move*.

Kedua algoritma diharapkan dapat mendeteksi serangan *copy-move* dengan berbagai karakteristik citra. Algoritma *robust match* juga diharapkan mendapatkan hasil yang lebih baik dibandingkan dengan *exact match* karena algoritma *exact match* memiliki fitur yang terbatas dimana hanya mendeteksi blok yang memiliki representasi piksel yang sama persis.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana algoritma *exact match* dapat mendeteksi serangan *copy-move*?
2. Bagaimana algoritma *exact match* menghasilkan hasil yang akurat?
3. Bagaimana algoritma *exact match* berjalan dengan waktu dan memori yang optimal?
4. Bagaimana algoritma r*obust match* dapat mendeteksi serangan *copy-move*?
5. Bagaimana algoritma *robust match* menghasilkan hasil yang akurat?
6. Bagaimana algoritma *robust match* berjalan dengan waktu dan memori yang optimal?
7. Pendekatan mana yang lebih baik dalam mendeteksi serangan *copy-move*?

1.3 Tujuan

Tujuan dari Tugas Akhir ini adalah mendeteksi serangan *copy-move* dengan menggunakan metode *block matching*.

1.4 Manfaat

Dengan dibuatnya Tugas Akhir ini maka manfaat yang diperoleh yaitu, mempermudah untuk mendeteksi serangan *copy-move* dalam kasus forensik digital maupun yang lainnya.

1.5 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Algoritma akan mendeteksi objek pada citra yang dilakukan *copy-move*, dimana *copy-move* menyalin suatu bagian lalu menempelkannya di bagian lain dalam satu citra yang sama.
2. Citra yang dideteksi merupakan citra hasil fotografi di mana setiap bagian memiliki ciri khas piksel. Citra yang akan dideteksi bukan berupa gambar polos yang hanya terdiri dari sedikit kombinasi piksel.
3. Citra yang akan dideteksi merupakan citra yang belum dilakukan kompresi setelah dilakukan *copy-move*.
4. Objek yang dilakukan *copy-move* merupakan objek yang serupa dengan objek asli, tidak diperbesar, diperkecil, dirotasi, maupun pengeditan lainnya.
5. Objek yang dideteksi dapat berupa apapun, termasuk *background*.
6. Implementasi algoritma dilakukan di lingkungan Python 2.7.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.

   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal Tugas Akhir tersebut berisi rencana Tugas Akhir yang akan dikerjakan sebagai syarat untuk menyelesaikan studi dan meraih gelar Strata-1 Teknik Informatika. Pada proposal tersebut dijelaskan secara garis besar tentang tahapan metode yang akan dilakukan.

2. Studi literatur

   Tahap ini merupakan tahap pengumpulan informasi yang diperlukan untuk pengerjaan Tugas Akhir sekaligus mempelajarinya. Beberapa literatur yang perlu dipelajari lebih dalam lagi untuk implementasi kedua metode, dan mengenai kompresi citra untuk mengimplementasikan algoritma *robust match*.

3. Analisis dan desain

   Pada tahap ini akan dilakukan analisis dan design perancangan aplikasi sesuai dengan tujuan yang dijabarkan. Kemudian disesuaikan dengan metode yang tepat, hal ini dimaksudkan agar nantinya ketika diimplementasikan ke dalam aplikasi dapat berjalan sesuai yang diharapkan.

4. Implementasi metode

   Kedua metode akan diimplementasikan dengan menggunakan bahasa pemrograman Python 2.7 dengan *library* utama OpenCV dan NumPy dengan fungsi yang sudah tersedia di dalamnya.

5. Pengujian dan evaluasi

   Metode akan diuji setelah selesai diimplementasikan menggunakan skenario yang sudah dipersiapkan. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian *output* dari algoritma dan jawaban yang

tersedia pada dataset. Serta dilakukan uji performa jalannya algoritma. Dengan melakukan pengujian dan evaluasi dimaksudkan juga untuk mengevaluasi jalannya program, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.

   Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam Tugas Akhir ini serta hasil dari implementasi yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini merupakan laporan secara lengkap mengenai Tugas Akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan Tugas Akhir. Selain itu, metodologi pengerjaan dan sistematika penulisan laporan Tugas Akhir juga terdapat di dalamnya.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Sistem

Bab ini berisi penjelasan tentang rancangan dari metode yang akan diimplementasikan. Terdapat dua tipe rancangan metode, yaitu rancangan yang berasal

dari *paper* rujukan dan rancangan pengembangan metode oleh penulis.

Bab IV Implementasi

Bab ini berisi penjelasan implementasi dari rancangan yang telah dibuat pada bab sebelumnya. Implementasi disajikan dalam bentuk kode sumber utama pada setiap metode disertai dengan penjelasannya.

Bab V Uji Coba Dan Evaluasi

Bab ini berisi penjelasan mengenai data hasil percobaan dan pembahasan mengenai hasil percobaan yang telah dilakukan.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan metode ke depannya.

BAB II
TINJAUAN PUSTAKA

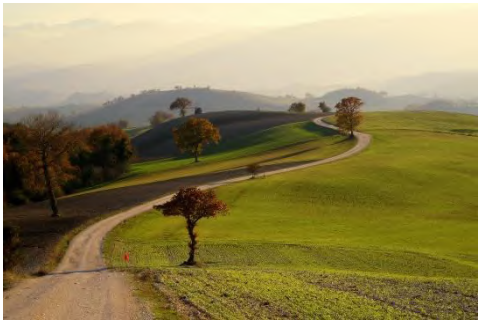
Dalam bab ini akan dijelaskan mengenai teori-teori yang merupakan dasar dari implementasi metode. Selain itu terdapat penjelasan yang menunjang pengerjaan Tugas Akhir ini sehingga dapat memberikan gambaran secara umum sistem yang akan dibangun.

2.1 Forensik Digital

Forensik digital adalah aktivitas yang berhubungan dengan pemeliharaan, identifikasi, ekstraksi dan dokumentasi bukti digital dalam kejahatan [2]. Tujuan dari forensik digital adalah menemukan, menganalisis, menjaga barang bukti pada artefak terkait. Bukti-bukti dalam forensik digital sangatlah banyak, seperti semua bukti fisik seperti komputer atau harddisk, dokumen-dokumen yang tersimpan dalam komputer seperti, gambar, PDF, video, lalu lintas data dalam jaringan, dan lain sebagainya. Dengan identifikasi barang bukti dapat memperkirakan efek yang terjadi, dan menemukan identitas pelaku yang mungkin. Karena luasnya cangkupan, forensik digital dibagi menjadi beberapa cabang seperti forensik komputer, forensik analisis data, forensik jaringan, forensik perangkat bergerak, atau forensik *database*..

2.2 Forensik Citra Digital

Forensik citra digital merupakan cabang dari keamanan multimedia yang bertujuan untuk memvalidasi keaslian suatu citra dengan mengetahui informasi riwayatnya. Pada Forensik citra digital kegiatan yang difokuskan yaitu mendeteksi pemalsuan citra. Contoh pemalsuan citra yang sering terjadi adalah menambahkan atau mengurangi informasi yang ada pada citra. [3]



Gambar 2.1 Contoh citra sebelum dilakukan serangan *copy-move*

Gambar 2.2 Contoh citra setelah dilakukan serangan *copy-move*

Untuk menambah informasi pada citra, dibutuhkan suatu informasi tambahan yang akan diletakkan pada citra. Untuk mengurangi informasi, biasanya pemalsu citra membutuhkan konten lain untuk menutupi informasi yang ingin dihilangkan. Pemalsuan seperti ini dapat terjadi dengan melibatkan satu atau lebih citra. Pemalsuan dengan melibatkan satu citra dapat disebut *copy-move*. Sedangkan pemalsuan dengan melibatkan lebih dari satu citra dapat disebut *copy-splice*. Gambar 2.1 merupakan citra asli di mana citra tersebut dilakukan serangan *copy-move*. Tantangan forensik digital adalah melakukan deteksi terhadap pemalsuan-pemalsuan yang terjadi.

2.3 *Copy-Move*

Copy-move adalah suatu cara di mana pengguna menyalin suatu bagian citra lalu menempelkannya di bagian lain pada citra yang sama. Teknik ini biasanya digunakan untuk menutupi objek dari suatu citra dengan cara menempelkan objek yang serupa di sekitarnya atau menempelkan objek pada posisi lain sehingga objek terlihat lebih banyak. Untuk menutupi objek, umumnya citra yang disalin adalah tekstur seperti rumput, dedaunan, kerikil, ataupun objek lain yang memiliki pola yang tidak beraturan tujuannya agar objek yang menutupi serupa dengan latar atau objek di sekitarnya sehingga sulit dibedakan dengan mata. [1] Gambar 2.2 merupakan contoh di mana pemalsu citra menyalin objek pohon dan menempelkannya pada posisi lain.

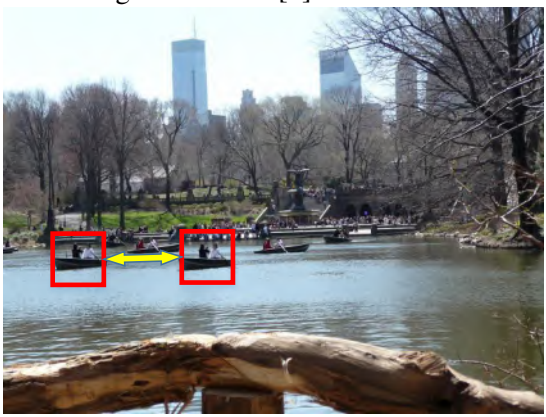
2.4 Citra Digital

Secara umum, pengolahan citra digital menunjuk pada pemrosesan gambar dua dimensi menggunakan komputer. Dalam konteks yang lebih luas, pengolahan citra digital mengacu pada pemrosesan setiap data dua dimensi. Citra digital merupakan sebuah larik (*array*) yang berisi nilai-nilai real maupun kompleks yang direpresentasikan dengan deretan bit tertentu. Suatu citra dapat didefinisikan sebagai fungsi $f(x,y)$ berukuran M baris dan N kolom, dengan x dan y adalah koordinat spasial, dan amplitudo f di titik koordinat (x,y) dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x, y, dan nilai amplitudo f secara keseluruhan berhingga (*finite*) dan bernilai diskrit maka dapat dikatakan bahwa citra tersebut adalah citra digital [4].

2.5 Metode *Block Matching*

Block matching dalam konteks citra untuk deteksi serangan *copy-move* adalah proses membagi citra menjadi beberapa bagian

persegi atau blok. Kemudian fitur pada setiap blok diekstrak untuk dapat dicocokkan dengan blok lain [5].



Gambar 2.3 Visualisasi *block matching*

Metode *block matching* memiliki banyak variasi dalam proses penerapannya. Variasi metode dapat berupa cara mendapatkan blok, fitur yang diambil dalam satu blok, maupun cara melakukan perbandingan antar fitur atau blok.

2.6 Python

Python adalah sebuah bahasa pemrograman *interpreter* yang dapat digunakan untuk berbagai keperluan. Dari sisi kualitas *software*, Python didesain untuk mudah dibaca, *reusable*, dan *maintainable*. Dari segi produktivitas, Python meningkatkan produktivitas pengembang berkali lipat dibandingkan dengan bahasa pemrograman *low level* seperti C, C++, dan Java. Python memiliki *library* yang dapat digunakan untuk menambah fitur-fitur sesuai dengan kebutuhan *software* yang akan dikembangkan. NumPy dan OpenCV merupakan salah satu *library* yang dapat digunakan untuk melakukan pengolahan citra digital [6].

2.7 OpenCV

OpenCV adalah *library open source* yang ditujukan untuk pengolahan visi komputer. *Library* ini ditulis dalam bahasa C dan C++ dan dapat berjalan pada *platform* Linux, Windows, dan MAC OS X. OpenCV memiliki beberapa *interface* yang dapat digunakan pada beberapa bahasa pemrograman, seperti Python, Ruby, Matlab, dan bahasa pemrograman lainnya [7].

2.8 NumPy

NumPy berasal dari kata *Numerical Python* merupakan *library* Python yang bersifat *open source* untuk pengolahan data ilmiah. Pada dasarnya NumPy bekerja dengan *array* dan matriks. Library ini berisi *list* yang dapat digunakan melakukan perhitungan fungsi matematika, seperti aljabar linear, *Fourier transform*, dan pembuatan bilangan acak [8].

2.9 Hash

Hash adalah sebuah nilai yang digunakan untuk mengecek integritas sebuah data. Hash juga bisa dikatakan enkripsi satu arah. Nilai hash didapatkan melalui fungsi hash. Fungsi hash adalah sebuah fungsi yang memiliki *input* bit *string* dengan panjang dinamis dan mengeluarkan bit *string* dengan panjang yang statis [9]. Hash biasanya digunakan dalam konteks keamanan informasi dan jaringan. Contoh penggunaan hash dalam konteks pengecekan integritas data adalah nilai hash pada situs yang menyediakan dokumen yang dapat diunduh oleh pengguna. Untuk mengetahui dokumen telah diunduh dengan sempurna, situs menyediakan nilai hash pada setiap dokumen. Apabila pengguna ingin mengecek apakah dokumen yang diunduh telah sesuai dengan yang ada pada situs tersebut, pengunduh dapat menghitung nilai hash pada dokumen yang diunduh, kemudian disamakan dengan nilai hash yang ada pada situs. Apabila nilai hash sama, maka dokumen yang diunduh sesuai dengan dokumen pada situs.

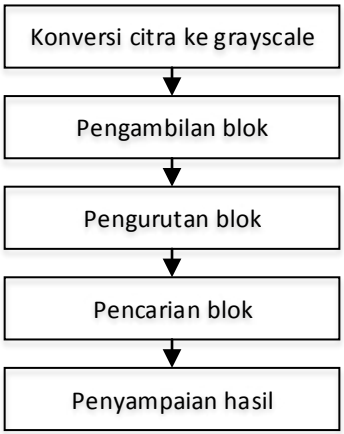
[Halaman ini sengaja dikosongkan]

BAB III
PERANCANGAN SISTEM

Pada bab ini akan dijelaskan mengenai perancangan sistem dari metode yang telah ada pada *paper* referensi [1] dan metode pengembangan yang dilakukan oleh penulis. Perancangan yang dijelaskan meliputi data dan proses. Data yang dimaksud adalah citra yang akan diolah ke dalam algoritma sehingga tujuan Tugas Akhir ini bisa tercapai.

3.1 Metode *Exact Match* Berdasarkan *Paper* Referensi

Metode *exact match* memiliki lima tahap utama, yaitu konversi citra ke *grayscale*, pengambilan blok, pengurutan blok, pencarian blok, dan penyampaian hasil.



Gambar 3.1 Tahap-tahap metode *Exact Match*

Pada tahap awal, citra dilakukan konversi ke dalam bentuk *grayscale*. Selanjutnya, blok adalah sebuah persegi berukuran *B*x*B* piksel. Pada tahap pengambilan blok, blok akan digeser setiap satu piksel pada semua bagian citra mulai dari pojok kiri atas sampai dengan pojok kanan bawah dengan

urutan ke kanan lalu ke bawah. Di setiap pergeseran posisi blok, nilai piksel pada blok tersebut diekstrak, kemudian disimpan pada *array A*. Di akhir iterasi *array A* akan memiliki baris sebanyak *(M-B+1)(N-B+1)* di mana *M* dan *N* adalah ukuran dari citra dan B^2. Setiap baris mencerminkan satu posisi dari pergeseran blok [1].

Dua baris yang identik di *array A*, menunjukkan kemiripan pada blok *BxB*. Untuk mengidentifikasi kemiripan blok, maka *array A* diurutkan secara leksikografi. Pengurutan secara leksikografi adalah pengurutan berdasarkan kamus. Blok diurutkan berdasarkan piksel paling kiri kemudian ke kanan. Pengurutan dapat dilakukan dengan kompleksitas mendekati $MNLog_2(MN)$. Dengan pengurutan ini dapat mempersingkat pencarian dibandingkan dengan melakukan perbandingan pada semua blok yang memiliki kompleksitas $(MN)^2$.

Pencarian dua blok yang sama dapat dilakukan dengan mudah apabila kumpulan blok sudah terurut. Pencarian blok yang sama dilakukan dengan cara melakukan iterasi sepanjang *array A*. Apabila terdapat dua blok yang sama maka posisi blok tersebut akan ditandai pada citra hasil.

Citra hasil merupakan citra dengan piksel berwarna hitam-putih. Piksel dengan warna putih berarti objek *copy-move*. Citra hasil kemudian diambil kontur atau keliling dari objek yang ditemukan untuk ditandai pada citra asli yang berwarna. Citra hasil hitam-putih dan citra berwarna yang telah ditandai inilah yang akan dikeluarkan ke pengguna.

3.2 Pengembangan Metode *Exact Match*

Metode *exact match* yang berasal dari *paper* referensi masih dapat dikembangkan kembali agar performa dari metode lebih baik. Pengembangan yang dilakukan pada metode *exact match* adalah penghapusan tahap konversi citra ke *grayscale*, tahap pengambilan blok dengan menggunakan nilai hash, dan

melakukan morfologi *opening* sebelum mengeluarkan hasil ke pengguna.

Gambar 3.2 Tahap-tahap hasil pengembangan metode *exact match*

Gambar 3.2 menggambarkan perubahan pengembangan metode *exact match* dari metode pada *paper* referensi. Berikut akan dijelaskan masing-masing tahap yang dilakukan perubahan.

3.2.1 Pemrosesan Citra dengan Bentuk RGB

Proses penghilangan konversi citra ke *grayscale* dilakukan untuk memperkecil kemungkinan kesalahan deteksi. Apabila proses dilakukan pada citra *grayscale*, kemungkinan nilai blok yang sama akan lebih besar. Proses konversi ke dalam bentuk grayscale pada *paper* referensi dilakukan untuk mempermudah perbandingan blok, karena apabila citra berbentuk RGB proses perbandingan dilakukan tiga kali pada masing-masing piksel, sedangkan *grayscale* hanya satu kali. Selain itu pemrosesan citra RGB juga akan mempersulit tahap pengurutan blok karena struktur data yang kompleks. Akan tetapi, pada pengembangan metode ini proses perbandingan struktur penyimpanan blok nilai hash yang akan dijelaskan pada

tahap selanjutnya. Sehingga pemrosesan citra dengan bentuk RGB dapat dilakukan pada metode pengembangan ini.

3.2.2 Penggunaan Nilai Hash

Pada tahap pengambilan blok, kompleksitas pengurutan secara leksikografi dengan struktur *array* dua dimensi sangatlah tinggi. Selain itu perbandingan antar baris *array* dua dimensi juga membutuhkan waktu yang lama. Untuk mempersingkat proses pengurutan dan perbandingan, struktur *array* diganti menjadi satu dimensi. Nilai yang disimpan pada array bukanlah piksel-piksel pada blok, melainkan nilai hash pada blok tersebut. Untuk memperoleh nilai hash pada blok, dapat menggunakan algoritma seperti SHA atau CRC. Selanjutnya pengurutan dan perbandingan blok dilakukan berdasarkan nilai hash blok tersebut. Penyimpanan nilai hash akan memperoleh beberapa keuntungan, yaitu penggunaan memori yang berjalan lebih sedikit, waktu pengurutan lebih cepat dan waktu perbandingan lebih cepat.

3.2.3 Morfologi *Opening*

Hasil pada metode *exact match* dari *paper* referensi masih menimbulkan beberapa kesalahan deteksi. Kesalahan deteksi berupa blok-blok kecil seperti pada Gambar 3.3 Citra hasil Uji Coba *giraffe.png* Skenario 2 dengan ukuran blok 8x8. Piksel dengan warna putih merupakan hasil deteksi *copy-move* sedangkan warna hitam tidak dideteksi sebagai serangan *copy-move*. Karena terdapat perbedaan ukuran objek yang signifikan pada citra hasil, objek-objek kecil dapat dilakukan penghapusan dengan cara melakukan operasi pengolahan citra yaitu morfologi *opening*.

Morfologi *opening* adalah salah satu cara untuk menghilangkan kesalahan-kesalahan kecil pada citra. *Opening* menghilangkan objek-objek kecil, sedangkan *closing* menutup lubang kecil.

Gambar 3.3 Citra hasil Uji Coba *giraffe.png* Skenario 2

Dengan melakukan morfologi *opening*, objek-objek kecil seperti pada Gambar 3.3 akan dihapus. Morfologi opening dilakukan pada tahap akhir sebelum pengeluaran citra hasil ke pengguna.

3.3 Metode *Robust Match* Berdasarkan *Paper* Referensi

Berbeda dengan metode *exact match* yang melakukan pencarian blok yang sama persis, metode *robust match* dapat mendeteksi blok yang memiliki nilai piksel yang hampir sama. Dengan menggunakan metode *robust match* diharapkan dapat meningkatkan kualitas dari hasil metode *exact match*. Peningkatan kualitas dapat berupa pendeteksian blok yang lebih banyak sehingga bentuk objek hasil ekstraksi dapat menyerupai bentuk objek pada citra asli.

Tahapan metode *robust match* umumnya hampir sama dengan metode *exact match*. Perbedaan utamanya adalah pada proses pengambilan blok. Proses pengambilan blok perlu dimodifikasi sehingga dapat mentoleransi sedikit perubahan nilai blok. Metode *robust match* dibagi menjadi beberapa tahap, yaitu konversi citra ke dalam bentuk *grayscale*, pengambilan blok dengan DCT dan kuantisasi, pengurutan blok, pencarian blok, dan penyampaian hasil [1].

Gambar 3.4 Alur metode *Robust Match*

Tahap awal dari metode *robust match* adalah konversi citra ke dalam bentuk *grayscale*. Tahap ini dilakukan agar citra dapat dilakukan perhitungan DCT (*Discrete cosine transform*) dan kuantisasinya pada tahap pengambilan blok.

Proses pengambilan blok tidak jauh berbeda dengan *exact match*. Akan tetapi pada metode *robust match* setiap blok dilakukan konversi piksel menjadi bentuk DCT lalu dihitung kuantisasinya. Nilai kuantisasi ini yang akan dimasukkan ke dalam *array A* yang akan dilakukan proses selanjutnya. Pada metode *robust match* blok berukuran 8x8, karena ukuran tersebut disesuaikan dengan matriks kuantisasi yang berukuran 8x8.

DCT dan kuantisasi digunakan untuk mendapatkan representasi blok yang *robust*. Berikut merupakan persamaan dari dua dimensi DCT-II:

$$G_{u,v} = \frac{1}{4}\,\alpha(u)\,\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

Di mana:

- u adalah horizontal, untuk *integer* $0 \leq u < 8$.
- v adalah vertikal, untuk *integer* $0 \leq v < 8$.
- $\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & jika\ u = 0 \\ 1, & selain\ u = 0 \end{cases}$ adalah faktor skala normalisasi untuk membuat transformasi *orthonormal.*
- $g_{x,y}$ adalah nilai piksel pada koordinat (x, y).
- $G_{u,v}$ adalah koefisien DCT pada koordinat (x, y)

Proses perhitungan kuantisasi adalah sebagai berikut :

$$C_{i,j} = round\left(\frac{D_{i,j}}{Q_{i,j}}\right)$$

Di mana:

- Q = matriks kuantisasi
- D = matriks hasil DCT
- C = hasil kuantisasi

Blok hasil kuantisasi dilakukan pengurutan secara leksikografi seperti pada metode *exact match*. Karena pada tahap pencarian yang dibandingkan adalah nilai kuantisasi dari DCT, metode ini akan menemui banyaknya kesalahan deteksi. Untuk mengurangi banyaknya kesalahan deteksi, algoritma mencari posisi yang berkaitan dari masing-masing pasangan blok yang sama. Untuk melakukan pencarian posisi yang berkaitan, jika terdapat dua blok yang berurutan pada *array A* dengan nilai yang sama, algoritma akan menyimpan posisi dari kedua blok pada *list* yang terpisah. Posisi yang disimpan dapat direpresentasikan posisi dari pojok kiri atas blok. Kemudian dilakukan penambahan pada *shift-vector counter C*. Perhitungan *shift-vector counter C* adalah sebagai berikut:

$$s = (s1, s2) = (i_1 - j_1, i_2 - j_2)$$
$$C(s1, s2) = C(s1, s2) + (i_1, i_2) + (j_1, j_2)$$

Di mana:

- (i_1, i_2) dan (j_1, j_2) adalah posisi dari dua blok yang sama
- s adalah *shift-vector*
- C adalah *shift-vector counter*

Shift-vector counter C selalu ditambahkan setiap menemui dua blok berurutan yang sama pada *array A*. Diakhir pencarian, algoritma akan dengan mudah menemukan *shift-vector* yang memiliki jumlah blok yang melebihi batas *threshold T* yang telah ditentukan pengguna.

Nilai *threshold T* memiliki hubungan dengan besar kecilnya objek yang dilakukan *copy-move*. Nilai *T* yang terlalu besar akan menyebabkan tidak terekstraknya objek *copy-move* sedangkan nilai yang terlalu kecil akan menghasilkan banyak kesalahan deteksi. Nilai *threshold T* dan ukuran blok *B* merupakan faktor yang mempengaruhi sensitifitas algoritma.

3.4 Pengembangan Metode *Robust Match*

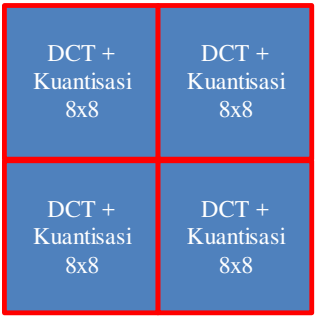
Metode *robust match* yang berasal dari *paper* referensi dapat dikembangkan lagi untuk mencapai hasil dan performa yang lebih baik. Pengembangan metode *robust match* dilakukan dengan menetapkan beberapa *threshold* tambahan dan pengambilan blok dengan ukuran yang lebih besar dari 8x8. Strutktur data pada pengembangan metode *robust match* sama seperti pengembangan metode *exact match*, yaitu menggunakan nilai hash.

3.4.1 Memperbesar Ukuran Blok

Salah satu cara untuk mencegah banyaknya kesalahan deteksi adalah dengan menggunakan ukuran blok yang lebih besar. Dengan ukuran blok yang lebih besar, dibutuhkan pula matriks kuantisasi yang lebih besar dari ukuran standar 8x8. Pada *paper* referensi metode *robust match* di bab sebelumnya, sudah menjelaskan perhitungan untuk mendapatkan matriks

kuantisasi dengan ukuran 16x16, akan tetapi dari *paper* disebutkan bahwa pemilihan matriks kuantisasi agak *ad hoc* dan tidak melakukan investigasi untuk pemilihan matriks kuantisasi lebih jauh, maka penulis memutuskan untuk melakukan perhitungan matriks dengan menggunakan metode yang berbeda.

Untuk mendapatkan ukuran blok yang lebih besar dari 8x8, perhitungan DCT dan kuantisasi tetap dilakukan dengan blok 8x8, kemudian hasil dari beberapa perhitungan blok 8x8 digabungkan dengan blok lain sehingga dapat menghasilkan blok yang lebih besar.



Gambar 4.1 Visualisasi blok

Gambar 4.1 Visualisasi blok menunjukkan perhitungan blok 16x16. Dengan perhitungan seperti ini, matriks kuantisasi standar dengan ukuran 8x8 tetap dapat dipergunakan.

3.4.2 *Threshold* Jarak Antar Blok

Threshold tambahan yang diterapkan pada metode ini adalah menghitung jarak antara dua blok berurutan yang bernilai sama yang ditemui pada *array A* saat proses pencarian blok. Penghitungan jarak dilakukan karena algroritma dapat mendeteksi blok yang bernilai hampir sama. Blok-blok yang berdekatan, kemungkinan besar bernilai hampir sama. Sehingga

blok-blok yang berdekatan dideteksi sebagai objek *copy-move* oleh algoritma.

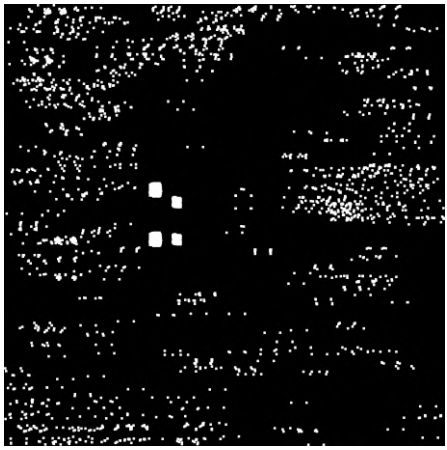
Gambar 3.5 Citra *giraffe.png* sebelum dan setelah deteksi

Gambar 3.5 menunjukkan banyaknya kesalahan deteksi (piksel berwarna putih) pada metode *robust match* dengan ukuran 8x8 menggunakan *threshold T* dengan nilai 400 piksel. Oleh karena itu dibutuhkan *threshold J*, yaitu jarak dari dua blok yang bernilai sama.

Threshold J dapat dilakukan pada tahap perhitungan *shift-vector* dengan cara tidak melakukan penambahan nilai *shift-vector counter C* apabila jarak dua blok tidak memenuhi *threshold*.

3.4.3 *Threshold* Banyak Objek Setiap *Shift-Vector*

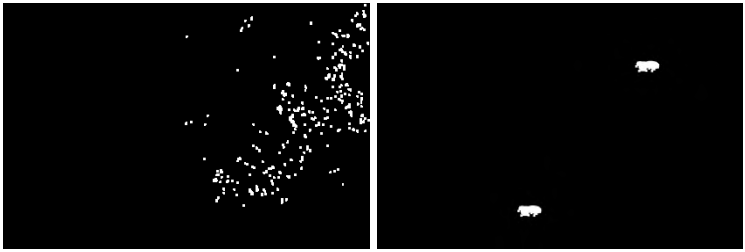
Pada beberapa kasus citra, posisi-posisi blok dengan *shift-vector* yang sama masih dapat menghasilkan kesalahan deteksi. Kesalahan deteksi terlihat dari posisi-posisi blok yang tersebar pada beberapa tempat. Seharusnya blok yang dideteksi sebagai objek *copy-move* berkumpul minimal pada dua titik. Dengan blok yang berkumpul pada minimal dua titik, blok-blok tersebut akan membentuk objek perkiraan *copy-move*. Akan tetapi apabila blok tersebar di berbagai citra, blok-blok tersebut tidak membentuk objek, sehingga tidak dapat dikatakan objek *copy-move*.



Gambar 3.6 Citra hasil uji coba *knight moves.png* skenario 4

Gambar 3.6 menunjukkan kesalahan deteksi masih terjadi meskipun sudah melalui tahap *threshold T* dan *threshold J* dengan ukuran blok 8x8. Maka dari itu, dibutuhkan *threshold K*, yaitu *threshold* dari banyaknya objek yang dihasilkan pada setiap *shift-vector*. Objek yang dimaksud adalah kumpulan blok yang saling menempel. Apabila objek terlalu banyak, maka dapat didikatakan bahwa blok pada *shift-vector* tersebut bukanlah blok *copy-move*.

Untuk menghitung banyaknya objek pada setiap *shift-vector*, posisi-posisi dalam satu *shift-vector* dikumpulkan ke dalam dua buah list. List pertama berisi semua posisi *y* dan list kedua berisi semua posisi *x*. Lalu kedua list diambil nilai uniknya dan dilakukan pengurutan. Kemudian masing-masing list dilakukan iterasi sambil menghitung perubahan nilai *x* atau *y*. Apabila perubahan nilai lebih dari satu, maka *counter obj* objek bertambah. *Counter* inilah yang akan dibandingkan dengan nilai *threshold K*. Perhitungan banyaknya objek dengan metode seperti ini tidaklah selalu benar pada semua kasus. Akan tetapi metode ini dapat menghasilkan hasil yang sebanding dengan waktu yang dibutuhkan.



Gambar 4.2 Perbedaan jumlah objek pada jarak tertentu

Gambar 4.2 menunjukkan perbedaan jumlah objek pada pada *shift-vector* yang berbeda. Gambar dikiri memiliki banyak objek sedangkan gambar di kanan hanya terdapat dua objek. Gambar di kiri tidak adakan ditampilkan karena dibatasi oleh *threshold K*.

BAB IV
IMPLEMENTASI

Setelah dijelaskan mengenai metode deteksi *copy-move* yang berasal dari *paper* referensi maupun metode pengembangan penulis pada Bab 3, pada Bab 4 ini akan dijelaskan mengenai implementasi terhadap metode yang telah dijelaskan.

4.1 Lingkungan Implementasi

Dalam proses implementasi deteksi *copy-move* pada citra digital diperlukan beberapa perangkat pendukung sebagai berikut.

4.1.1. Lingkungan Implementasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan pada lingkungan pengembangan aplikasi adalah sebagai sebuah komputer PC atau laptop sebagai komputer pemroses dengan spesifikasi:

- Prosesor : AMD A6-4455M APU 2.1 GHz
- Memori : 6.00 GB
- Tipe Sistem : 64-bit *Operating System*

4.1.2. Lingkungan Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan pada lingkungan pengembangan aplikasi adalah sebagai berikut:

- Windows 8.1 sebagai sistem operasi
- Python 2.7 dengan *library* utama OpenCV dan NumPy dengan IDE PyCharm untuk mengimplementasikan metode.

4.2 Implementasi Metode *Exact Match*

Pada sub bab ini dijelaskan mengenai fungsi-fungsi pada implementasi metode *exact match* sesuai pada Bab III. Metode *exact match* langsung diimplementasikan dengan menggunakan penyimpanan blok bebentuk nilai hash mengingat besarnya dataset yang digunakan. Sedangkan hal lainnya sesuai dengan *paper* referensi.

4.2.1 Konversi Citra Menjadi *Grayscale*

Proses konversi citra menjadi bentuk *grayscale* dilakukan pada fungsi utama karena *syntax* untuk melakukan konversi RGB ke *grayscale* cukup dengan satu perintah dengan menggunakan OpenCV.

```
1   grayscale_image = cv2.cvtColor(rgb_image,
                                   cv2.COLOR_BGR2GRAY)
```

Kode Sumber 4.1 Konversi citra RGB ke *grayscale*

Fungsi *cvtColor* merupakan fungsi untuk melakukan konversi warna pada citra. Fungsi tersebut membutuhkan dua parameter, yaitu image masukan dan keterangan konversi. *rgb_image* merupakan variabel yang menampung citra dalam bentuk RGB sedangkan *COLOR_BGR2GRAY* berarti fungsi akan melakukan konversi bentuk RGB ke *grayscale*.

4.2.2 Implementasi proses pengambilan blok

Proses pengambilan blok dijadikan dalam sebuah fungsi. Data masukan berupa citra dan ukuran blok. Citra masukan dapat berupa citra RGB maupun *grayscale*. Ukuran blok terdiri dari satu bilangan yang mencerminkan blok *NxN*.

```
1   def get_blocks(img, block_size):
2       img_height, img_width = img.shape[0:2]
```

```
3       blocks = []
4       for y in range(0, img_height -
    block_size + 1):
5           for x in range(0, img_width -
              block_size + 1):
6               block = img[y: y + block_size,
                      x: x+block_size]
7               block = block.copy(order='C')
8               block_hash =
                      binascii.crc32(block)
9               block = str(block_hash) + '_' +
                      str(x) + '_' + str(y)
10              blocks.append(block)
11      return blocks
```

Kode Sumber 4.2 Pengambilan blok *exact match*

Proses pengambilan blok dilakukan dengan cara melakukan pergeseran blok kekanan lalu kebawah sepanjang ukuran citra yang dimulai dari pojok kiri atas. Pada Kode Sumber 4.2 pergeseran blok ditunjukan pada perulangan di baris keempat dan kelima. Setiap pergerseran blok diambil piksel yang berada didalam blok tersebut, disimpan pada variabel *block* pada baris keenam. Nilai hash dari *block* dihitung dengan menggunakan algoritma CRC32 pada baris ketujuh. Nilai hash digunakan untuk mempercepat proses perhitungan selanjutanya dan untuk menghemat penggunaan memori. Hash yang didapat lalu dikonversi menjadi string kemudian disambung dengan posisi *x* dan posisi *y* dari blok tersebut. Blok disimpan dalam sebuah string yang berisi hash piksel, posisi *x*, dan posisi *y* dipisahkan dengan karakter *underscore*. Penyimpanan dalam bentuk string dapat menghemat penggunaan memori yang berjalan dibandingkan dengan penyimpanan dalam bentuk *list* ataupun *dictionary*.

4.2.3 Implementasi Proses Pengurutan

Pengurutan blok dilakukan untuk mempermudah proses pencarian sehingga pencarian blok dapat berjalan lebih cepat. Karena blok bertipe data string, maka pengurutan akan dilakukan berdasarkan urutan karakter mulai dari yang terdepan. Berdasarkan penyimpanan blok pada proses sebelumnya, nilai pertama yang ada pada blok adalah nilai hash. Sehingga blok akan diurutkan berdasaran nilai hash.

```
1  blocks = sorted(blocks)
```

Kode Sumber 4.3 Pengurutan blok

Pada Kode Sumber 4.3 pengurutan dilakukan hanya menggunakan satu perintah. Fungsi *sorted* akan melakukan pengurutan pada sebuah variabel yang dapat diiterasi, seperti *list*, *string*, *tuple*. Secara *default* pengurutan akan dilakukan secara leksikografi, atau urut abjad.

4.2.4 Implementasi Pencarian Blok

Setelah dilakukan pengurutan, blok akan dilakukan pencarian. Pencarian dilakukan dengan cara mengiterasi blok yang sudah terurut. Apablia ditemui dua blok yang memiliki nilai hash yang sama, posisi dari blok tersebut akan ditandai pada citra hasil pengolahan.

```
1  def search_and_mark(blocks, result_image,
                       block_size):
2      marker_image = np.zeros((block_size,
                     block_size, 3), np.uint8)
3      marker_image[:] = [255, 255, 255]

4      for currentIndex, block1 in
                       enumerate(blocks[:-1]):
5          block2 = blocks[currentIndex+1]
6          block2 = block2.split('_')
7          block1 = block1.split('_')
```

```
8           if block1[0] == block2[0]:
9               pos_y = int(block1[2])
10              pos_x = int(block1[1])
11              result_image[pos_y:
                    pos_y+block_size,
                  pos_x: pos_x+block_size] =
                  marker_image

12              pos_y = int(block2[2])
13              pos_x = int(block2[1])
14              result_image[pos_y:
                  pos_y+block_size,
                  pos_x: pos_x+block_size] =
                  marker_image

15      return result_image
```

Kode Sumber 4.4 Pencarian blok *exact match*

Pada Kode Sumber 4.4 fungsi *search_and_mark* memiliki tiga parameter, yaitu *blocks* list dari kumpulan blok, *result_image* variabel yang menampung hasil posisi objek *copy-move*, dan *block_size* ukuran blok. Pada baris kedua dan ketiga, dibuat sebuah variabel untuk mendandai *result_image* sesuai dengan ukuran blok. Pada baris keempat, dilakukan iterasi semua blok yang ada. Setiap iterasi, diambil nilai hash dari dua blok yang berurutan pada baris kelima sampai ketujuh. Apabila nilai hash sama, *result_image* dilakukan penandaan dengan *mark_image* sesuai dengan posisi blok. Setelah iterasi selesai, fungsi mengembalikan *result_image*.

4.2.5 Implementasi Penyampaian Hasil

Tahap terakhir dari metode *exact match* yaitu penyampaian hasil. Terdapat dua keluaran dari metode ini, yang pertama adalah citra yang memiliki latar hitam dan menampilkan objek *copy-move*, yang kedua adalah citra asal yang ditempelkan garis yang menunjukkan bahwa garis tersebut adalah objek *copy-move*.

Untuk mendapatkan garis keliling dari citra hasil pengolahan, digunakan fungsi *findContours* pada library OpenCV.

```
1   def write_contours(source_img, dest_img):
2       source_img_gray = cv2.cvtColor(source_img,
           cv2.COLOR_BGR2GRAY)
3       ret, thresh=cv2.threshold(source_img_gray,
           127, 255, 0)
4       contours, hierarchy =
            cv2.findContours(thresh,
            cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
5       cv2.drawContours(dest_img, contours, -1,
            (0, 0, 255), 3)
6       return dest_img


7   def write_result(result_image, rgb_path):
8       dest_path = 'result\\bw_' +
          rgb_path.split('\\')[1]
9       cv2.imwrite(dest_path, result_image,
          [int(cv2.IMWRITE_PNG_COMPRESSION), 0])

        rgb_image = cv2.imread(rgb_path)
10      rgb_image = write_contours(result_image,
11        rgb_image)

        dest_path = 'result\\rgb_' +
12        rgb_path.split('\\')[1]
        cv2.imwrite(dest_path, rgb_image,
13        [int(cv2.IMWRITE_PNG_COMPRESSION), 0])
```

Kode Sumber 4.5 Penyampaian hasil

Pada Kode Sumber 4.5 terdapat dua fungsi, *write_contour* dan *write_result*. Fungsi *write_contour* akan mewarnai objek *copy-move* pada citra RGB berdasarkan citra hitam-putih. Baris kedua dan ketiga melakukan konversi citra ke RGB agar sesuai dengan kebutuhan fungsi *findContour*. Baris keempat menjalankan fungsi *findCountour*. Baris kelima

menandai citra RGB sesuai kontur yang didapatkan pada fungsi *findContour*. Fungsi *write_result* melakukan penyimpanan citra hasil olahan ke dalam komputer.

4.2.6 Implementasi Fungsi Utama

Fungsi utama (*main*) merangkum tahap-tahap yang dilakukan pada metode *exact match*. Berikut adalah kode sumber fungsi utama.

```
1  rgb_image = cv2.imread(imagePath)
2  grayscale_image = cv2.cvtColor(rgb_image,
                     cv2.COLOR_BGR2GRAY)
3  imageHeight, imageWidth =
                    grayscale_image.shape

4  blocks = get_blocks(grayscale_image,
           block_size)
5  del grayscale_image

6  blocks = sorted(blocks)

7  resultImage = np.zeros((imageHeight,
                imageWidth, 3), np.uint8)

8  resultImage = search_and_mark(blocks,
                resultImage, block_size)

9  write_result(resultImage, imagePath)
```

Kode Sumber 4.6 Fungsi utama

Pada Kode Sumber 4.6 baris pertama membaca citra yang telah dilakukan serangan *copy-move*. Baris kedua melakukan konversi RGB ke *grayscale*. Baris keempat melakukan pengambilan blok. Baris ke enam melakukan pengurutan blok. Baris kedelapan melakukan pencarian dan penandaan posisi. Baris kesepuluh menyimpan citra hasil ke komputer.

4.3 Implementasi Pengembangan Metode *Exact Match*

Pada sub bab ini dijelaskan mengenai fungsi-fungsi yang digunakan untuk pengembangan metode *exact match*. Terdapat tiga hal perubahan pada pengembangan *exact match*. Yang pertama ialah pemrosesan citra dengan bentuk RGB. Perubahan ini berpengaruh pada fungsi utama. Yang kedua penggunaan nilai hash yang telah diimplementasikan pada metode *exact match* yang sesuai dengan *paper* referensi. Sedangkan yang terakhir ialah morfologi *opening* hasil.

4.3.1 Implementasi Morfologi *Opening*

Morfologi *opening* merupakan proses pengolahan citra digital yang bertujuan untuk menghilangkan *noise* atau objek kecil pada citra. Morfologi *opening* dapat digunakan untuk menghilangkan kesalahan deteksi yang menimbulkan munculnya beberapa objek kecil.

```
1  def morph_open_image(result_image):
2      kernel = np.ones((20, 20), np.uint8)
3      result_image =
          cv2.morphologyEx(result_image,
          cv2.MORPH_OPEN, kernel)
4      return result_image
```

Kode Sumber 4.7 Morfologi *opening*

Pada Kode Sumber 4.7 baris kedua mendefinisikan sebuah *kernel* atau penutup yang digunakan untuk menutup objek kecil. Besar kecilnya objek yang akan ditutup sesuai dengan *kernel* yang digunakan. Pada Tugas Akhir ini digunakan ukuran 20x20 piksel. Baris ketiga melakukan morfologi citra *opening* sesuai dengan citra masukan dan kernel yang *diinputkan*.

4.3.2 Implementasi Fungsi Utama

Fungsi utama (*main*) tidak jauh berbeda dengan implementasi pada Sub Bab 4.2. Yang membedakan adalah tidak adanya konversi RGB ke grayscale dan dijalankannya fungsi morfologi *opening*.

```
1  image = cv2.imread(imagePath)
2  imageHeight, imageWidth, imageChannels =
   image.shape

3  blocks = get_blocks(image, block_size)
4  blocks = sorted(blocks)

5  resultImage = np.zeros((imageHeight,
   imageWidth, 3), np.uint8)

6  resultImage = search_and_mark(blocks,
   resultImage, block_size)

7  resultImage = morph_open_image(resultImage)

8  write_result(resultImage, imagePath)
```

Kode Sumber 4.8 Fungsi utama pengembangan metode *exact match*

Pada Kode Sumber 4.8 baris pertama membaca citra yang telah dilakukan serangan *copy-move*. Baris ketiga melakukan pengambilan blok. Baris ke empat melakukan pengurutan blok. Baris keenam melakukan pencarian dan penandaan posisi. Baris ketujuh melakukan morfologi *opening* citra hasil. Baris kedelapan menyimpan citra hasil ke komputer.

4.4 Implementasi Metode *Robust Match*

Pada sub bab ini dijelaskan mengenai fungsi-fungsi pada implementasi metode *robust match* sesuai dengan *paper*

referensi. Fungsi-fungsi yang dijelaskan pada sub bab ini merupakan fungsi yang berbeda dengan metode *exact match*.

4.4.1 Implementasi Proses Pengambilan Blok

Tahap pengambilan blok, dijadikan sebuah fungsi yang menerima argumen citra *grayscale* dan mengeluarkan *list* yang berisi nilai hash blok yang didapatkan.

```
1   def get_blocks(img):
2       blocks = []
3       for y in range(0,iHeight-block_size+1):
4           for x in range(0,iWidth-block_size+1):
5               block = img[y:y+block_size,
                  x:x+block_size]
6               dst = cv2.dct(block)
7               block_q =
8                 np.around(np.divide(dst,
                  std_luminance_quant_tbl))
                block_hash =
9                 binascii.crc32(block_q)
                blocks.append((block_hash, y, x))
10      return blocks
11
```

Kode Sumber 4.9 Pengambilan blok *robust match*

Proses pengambilan blok dilakukan dengan cara melakukan pergeseran blok ke kanan lalu ke bawah sepanjang ukuran citra yang dimulai dari pojok kiri atas. Pada Kode Sumber 4.9 pergeseran blok di tunjukan pada perulangan di baris ketiga dan keempat. Setiap pergeseran blok diambil piksel yang berada di dalam blok tersebut, disimpan pada variabel *block* pada baris kelima. Nilai DCT dari blok dihitung, kemudian dibagi dengan matriks kuantisasi pada baris kedelapan. Setelah itu nilai hash dari kuantisasi dihitung dengan menggunakan algoritma CRC32 pada baris kesembilan. Nilai hash digunakan untuk mempercepat proses perhitungan selanjutanya dan untuk menghemat penggunaan memori. Hash yang didapat disimpan ke dalam variable bertipe *tuple* beserta

dengan posisi *x* dan posisi *y* dari blok tersebut pada baris kesembilan. Variabel *tuple* disimpan ke dalam list *blocks* untuk proses selanjutnya.

4.4.2 Implementasi Proses Pencarian Blok

Pada tahap ini akan dilakukan pencarian blok yang sama. Tahap ini dilakukan dengan cara mengiterasi list blocks dan melakukan pengecekan setiap dua blok. Apabila nilai hash blok sama, maka di kedua blok tersebut dicari jarak posisinya (*shift-vector)*. Kemudian menyimpan posisi tersebut pada sebuah *list C*. *List C* berlaku sebagai *shift-vector counter* maupun list yang menampung posisi objek.

```
1   def matching(blocks):
2       c = [[[] for x in range(iWidth)] for y in
             range(iHeight)]
3       for idx, block in enumerate(blocks[:-1]):
            # block => (hash,y,x)
4           block1 = blocks[idx]
5           block2 = blocks[idx+1]
6           if block1[0] == block2[0]:
7               distance_y = abs(block2[1] -
                             block1[1])
8               distance_x = abs(block2[2] -
                             block1[2])
9               c[distance_y]
                 [distance_x].append(block1[1:3])
10              c[distance_y]
                 [distance_x].append(block2[1:3])
11      return c
```

Kode Sumber 4.10 Pencarian blok *robust match*

Pada Kode Sumber 4.10 fungsi *matching* memiliki argumen *blocks*, yaitu sebuah *list* yang berisi nilai hash, dan posisi setiap blok. Baris kedua fungsi menginisiasi *shift-vector counter* yang bertipe data *list of list* sebanyak ukuran citra. Baris ketiga fungsi melakukan iterasi pada semua blok yang ada,

kemudian membandingkan nilai hash dua blok yang berurutan di baris keenam. Apabila nilai hash sama, *shift-vector* dihitung pada baris ketujuh dan kedelapan. Lalu *shift-vector counter* ditambahkan posisi dua blok yang sama pada baris kesembilan dan sepuluh. Fungsi *matching* mengeluarkan hasil variabel *c*, yaitu *list* posisi dari blok yang memiliki nilai hash yang sama.

4.4.3 Implementasi *Thresholding*

Hasil pencarian berupa variabel *c* yang berisi posisi-posisi ojek yang sama yang dikelompokkan berdasarkan *shift-vector* yang sama. Variabel *c* masih dapat menghasilkan hasil yang salah, oleh karena itu dibutuhkan beberapa proses *thresholding* untuk memastikan bahwa posisi tersebut adalah objek dari *copy-move*. *Thresholding* dilakukan dengan cara menghitung banyaknya posisi pada masing-masing *shift-vector*.

```
1  def marking(c):
2      global block_size
3      global mark_image

4      for selisihX in c:
5          for s2 in selisihX:
6              if len(s2) <= threshold_T:
7                  continue
8              for posisi in s2:
9                  pos_x = posisi[1]
10                 pos_y = posisi[0]
11                 result_image[pos_y:
                              pos_y+block_size,
                              pos_x:
                              pos_x+block_size] =
                              mark_image
12     cv2.imwrite('result.png',
                   result_image)
```

Kode Sumber 4.11 *Thresholding T*

Fungsi pada Kode Sumber 4.11 memiliki parameter *c*, yaitu *shift-vector counter*. Fungsi akan mengiterasi semua elemen pada variabel *c*, pada baris keempat dan kelima. Baris keenam fungsi melakukan *thresholding* banyaknya posisi pada setiap *shift-vector*. Apabila tidak memenuhi nilai *threshold*, maka fungsi tidak akan memprosesnya lebih lanjut. Apabila memenuhi nilai *threshold*, fungsi akan menandai citra hasil dengan semua posisi pada *shift-vector* tersebut.

4.5 Implementasi Pengembangan Metode *Robust Match*

Pada sub bab ini dijelaskan mengenai fungsi-fungsi yang digunakan untuk pengembangan metode *robust match*. Terdapat tiga hal perubahan pada pengembangan *robust match*. Pengembangan pertama ialah metode untuk memperbesar ukuran blok, yang kedua penggunaan *threshold* jarak antar blok, dan yang terakhir penggunaan *threshold* banyaknya objek setiap *shift-vector*.

4.5.1 Implementasi Metode Perbesaran Ukuran Blok

Salah satu cara untuk mengurangi kesalahan deteksi pada metode *robust match* adalah dengan memperbesar ukuran blok. Akan tetapi ukuran blok pada metode *robust match* harus sama dengan matriks kuantisasi. Agar tidak mengubah matriks kuantisasi yang standar, perhitungan tetap menggunakan ukuran blok 8x8. Perbesaran ukuran blok dilakukan dengan menggabungkan beberapa perhitungan blok dengan ukuran standar.

```
1  def get_blocks(img):
2      blocks = []
3      for y in range(0,iHeight-block_size+1):
4          for x in range(0,iWidth-block_size+1):
5              block = []
6              for i in range(block_size/8):
7                  for j in range(block_size/8):
```

```
8                   tmp_y = y + i*8
9                   tmp_x = x + j*8
                    sub_block =
                     img[tmp_y:tmp_y+8,
                     tmp_x:tmp_x+8]
11                  dst = cv2.dct(sub_block)
12                  sub_block_q =
                     np.around(np.divide(dst,
                     std_luminance_quant_tbl))
13                  block = np.append(block,
                            sub_block_q)

14          blocks.append((get_hash(block),
                           y, x))
15      return blocks
```

Kode Sumber 4.12 Perbesaran ukuran blok

Perbedaan utama dengan Kode Sumber 4.12 dengan Kode Sumber 4.2 adalah pada setiap pergeseran, terdapat iterasi yang berguna untuk pengambilan blok ukuran 8x8 pada baris keenam dan ketujuh. Pada baris 8 sampai 12, proses pengambilan blok dengan menghitung DCT dan kuantisasinya. Di akhir iterasi, blok ukuran 8x8 digabungkan dengan blok yang lebih besar pada baris 13.

4.5.2 Implementasi Threshold Jarak Antar Blok

Karena metode *robust match* dapat mendeteksi objek yang hampir sama, metode akan mendeteksi banyak objek dengan jarak yang dekat. Oleh karena itu, dibutuhkan nilai *threshold* yang membatasi jarak antar dua blok yang dideteksi. Penerapan *threshold* ini dilakukan pada saat proses pencarian blok.

```
1   def matching(blocks):
2       c = [[[] for x in range(iWidth)] for y in
            range(iHeight)]
3       for idx, block in enumerate(blocks[:-1]):
```

```
            # block => (hash,y,x)
4           block1 = blocks[idx]
5           block2 = blocks[idx+1]
6           if block1[0] == block2[0]:
7               distance_y = abs(block2[1] -
                             block1[1])
8               distance_x = abs(block2[2] -
                             block1[2])
9               if distance_x >= threshold_J
                  or distance_y >= threshold_J:
                    c[distance_y][distance_x].
                     append(block1[1:3])
10                  c[distance_y][distance_x].
                     append(block2[1:3])
11      return c
```

Kode Sumber 4.13 *Threshold* Jarak Antar Blok

Kode Sumber 4.10 memiliki sedikit perbedaan dengan Kode Sumber 4.13, yaitu Kode Sumber 4.13 memiliki perbandingan pada baris sembilan bahwa jarak antar dua blok harus melebihi *threshold J* sebelum memasukkan posisi ke dalam *shift-vector counter*.

4.5.3 Implementasi *Threshold* Banyak Objek Setiap *Shift-Vector*

Untuk lebih memperketat hasil deteksi, diterapkan nilai *threshold* yang dapat membatasi banyaknya objek pada setiap *shift-vector*. Sebab, apabila sebuah *shift-vector* berisi banyak objek, kemungkinan besar objek-objek tersebut adalah kesalahan deteksi. Perhitungan banyaknya objek dilakukan setelah *thresholding T* atau pada tahap *thresholding*.

```
1   def marking(c):
2       global block_size
3       global mark_image

4       for selisihX in c:
5           for s2 in selisihX:
6               if len(s2) <= threshold_T:
7                   continue
8               if is_obj_greater_than(s2,
                                threshold_K):
9                   continue
10              for posisi in s2:
11                  pos_x = posisi[1]
12                  pos_y = posisi[0]
13                  result_image[pos_y:
                                pos_y+block_size,
                                pos_x:
                                pos_x+block_size] =
                                mark_image
12      cv2.imwrite('result.png',
                    result_image)
```

Kode Sumber 4.14 Pemanggilan fungsi *thershold_K*

Fungsi *marking* pada Kode Sumber 4.14 tidak jauh berbeda pada dengan dengan fungsi pada sub bab 4.4.3. Perbedaannya adalah fungsi ini melakukan *threshold* pada baris kedelapan dengan cara melakukan pemanggilan fungsi *si_obj_greater_than* yang memiliki parameter *s2*, yaitu list posisi pada satu *shift-vector* dan nilai *threshold_K* yaitu maksimal banyaknya objek yang ditemui.

```
1   def is_obj_greater_than(obj, max_object):
2       only_x = zip(*obj)[1]
3       only_y = zip(*obj)[0]
4       only_x = tuple(set(only_x))
5       only_y = tuple(set(only_y))
6       sorted_x = sorted(only_x)
7       sorted_y = sorted(only_y)

8       global iHeight
9       global iWidth
10      if iWidth >= iHeight:
```

```
11          obj_counter = 1
12          for idx, x in enumerate(
                        sorted_x[:-1]):
13              if sorted_x[idx+1] -
                   sorted_x[idx] > 1:
14                  obj_counter += 1
15                  if obj_counter > max_object:
16                      return True
17      else:
18          obj_counter = 1
19          for idx, x in enumerate(
                        sorted_y[:-1]):
20              if sorted_y[idx+1] -
                   sorted_y[idx] > 1:
21                  obj_counter += 1
22                  if obj_counter > max_object:
23                      return True
24      return False
```

Kode Sumber 4.15 Fungsi *threshold_K*

Pada Kode Sumber 4.15 baris kedua sampai ketujuh bertujuan untuk melakukan pemisahan posisi *x* dan *y* kemudian mengambil nilai unik dari masing-masing posisi dan melakukan pengurutan posisi. Pemrosesan dilakukan berdasarkan posisi x atau y sesuai dengan panjang atau lebarnya citra. Algoritma melakukan iterasi pada setiap posisi di baris 12 atau 19. Kemudian melakukan pengecekan, apabila selisih 2 posisi lebih dari satu, maka *counter* ditambahkan. Nilai *counter* selalu dilakukan pengecekan apabila *counter* melebihi *threshold*, maka algoritma mengeluarkan nilai *True*. Di baris terakhir algoritma mengeluarkan nilai *False* yang berarti sesuai *threshold*.

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab kelima ini akan dijelaskan mengenai skenario dan uji coba dari metode yang telah diimplementasikan. Selain itu, hasil uji coba akan dievaluasi kinerjanya sehingga dapat diambil kesimpulan apakah metode yang berasal dari *paper* referensi maupun metode pengembangan penulis mampu menyelesaikan permasalahan yang telah dirumuskan di awal.

5.1 Lingkungan Uji Coba

Sebelumnya, perlu diketahui lingkungan uji coba perangkat keras maupun perangkat lunak yang digunakan pada uji coba Tugas Akhir ini. Lingkungan tersebut ditunjukkan pada Tabel 5.1 berikut ini.

Tabel 5.1 Spesifikasi lingkungan uji coba

Perangkat	Spesifikasi
Perangkat keras	AMD A6-4455M APU 2.1 GHz Memori: 6.00 GB
Perangkat lunak	Sistem Operasi: Microsoft Windows 8.1 64-bit Perangkat Pengembang: Python 2.7 dengan library utama OpenCV dan NumPy

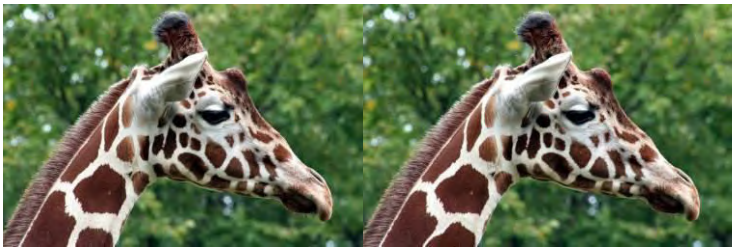
5.2 Data Uji Coba

Data uji coba yang digunakan pada Tugas Akhir ini adalah *Image Manipulation Dataset* [5]. Dataset ini didapatkan dari internet yaitu https://www5.cs.fau.de/research/data/image-manipulation/ [10] di akses pada tanggal 27 April 2016.

Diambil sembilan buah citra sebagai sampel dataset yang akan dilakukan uji coba, yaitu pada Tabel 5.2.

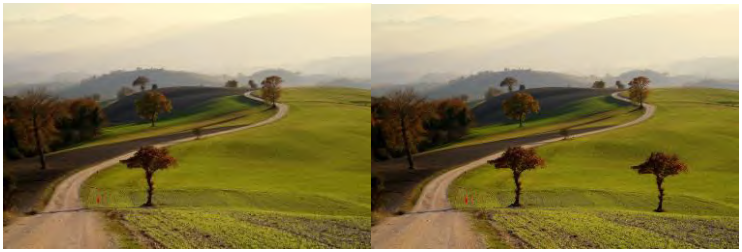
Nama File	Ukuran Piksel
giraffe.png	426,400
tree.png	699,392
clean walls.png	772,096
cattle.png	1,093,120
knight moves.png	2,073,600
threehundred.png	3,734,322
statue.png	5,596,640
malawi.png	7,077,888
scotland.png	7,077,888

Tabel 5.2 Citra uji coba



Gambar 5.1 *giraffe.png* sebelum dan setelah *copy-move*

Gambar 5.2 *Ground truth giraffe.png*



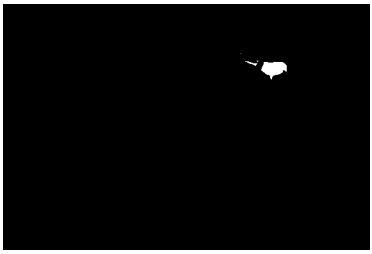
Gambar 5.3 ***tree.png*** **sebelum dan sesudah** ***copy-move***

Gambar 5.4 ***Ground truth tree.png***

Gambar 5.5 ***clean walls.png*** **sebelum dan sesudah** ***copy-move***

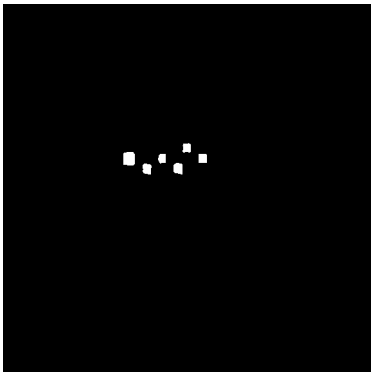
Gambar 5.6 ***Ground truth clean walls.png***

Gambar 5.7 ***cattle.png*** **sebelum dan sesudah** ***copy-move***



Gambar 5.8 ***Ground truth cattle.png***

Gambar 5.9 ***knight moves.png*** **sebelum dan sesudah** ***copy-move***

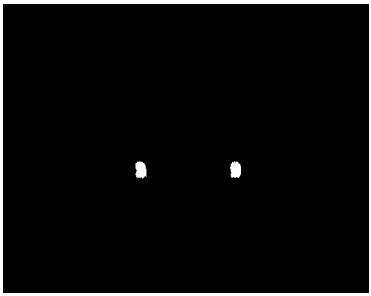


Gambar 5.10 ***Ground truth knight moves.png***

Gambar 5.11 ***threehundred.png*** **sebelum dan sesudah** ***copy-move***

Gambar 5.12 ***Ground truth threehundred.png***

Gambar 5.13 ***statue.png*** **sebelum dan sesudah** ***copy-move***



Gambar 5.14 ***Ground truth statue.png***

Gambar 5.15 ***malawi.png*** **sebelum dan sesudah** ***copy-move***

Gambar 5.16 ***Ground truth malawi.png***

Gambar 5.17 ***scotland.png*** **sebelum dan sesudah** ***copy-move***

Gambar 5.18 ***Ground truth scotland.png***

Sembilan buah citra terdiri dari berbagai ukuran dan ciri khas citra maupun objek yang dilakukan *copy-move*. Pada gambar Gambar 5.1 sampai dengan Gambar 5.17 merupakan citra yang akan digunakan pada uji coba. Citra di kiri merupakan citra asli sebelum dilakukan serangan *copy-move*. Sedangkan citra di kanan sudah dilakukan serangan *copy-move*.

5.3 Metode Evaluasi

Dibutuhkan suatu acuan untuk mengetahui performa dari metode dan uji coba yang telah dilakukan. Acuan yang digunakan yaitu menghitung *precision* dan *recall* dari hasil metode dan *ground truth* pada dataset. *Precision* menunjukkan bahwa objek hasil deteksi adalah sesuai dengan objek serangan. Sedangkan *recall* menunjukkan kemungkinan objek serangan yang terdeteksi. Perhitungan *precision* dan recall dilakukan di tingkat piksel. [5]

Untuk mendapatkan nilai *precision* dan *recall*, citra hasil pengolahan dibandingkan dengan ground truth. Kemudian dicari nilai *TP, TN, FP, FN*. *True Positives* (*TP*) merupakan suatu kondisi dimana prediksi bernilai benar, dan hasil sesungguhnya juga bernilai benar. *True Negatives* (*TN*) adalah kondisi dimana prediksi bernilai salah dan yang sebenarnya bernilai salah. *False Positives* (*FP*) merupakan kondisi dimana nilai prediksi adalah benar dan sebenarnya adalah salah (*Type I error*), dan *False Negatives* (FN) adalah kondisi dimana prediksi bernilai salah dan sebenarnya bernilai benar (*Type II error*). Dari nilai-nilai tersebut, dapat diperoleh *precision* dan recall dengan rumus sebagai berikut:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Dapat dihitung pula nilai F1 yaitu kombinasi dari *precision* dan *recall* untuk menghasilkan sebuah nilai tunggal.

$$F1 = \frac{2.Precision\ .Recall}{Precision\ +\ Recall}$$

Pada setiap skenario akan ditampilkan nilai-nilai yang telah dijelaskan beserta penjelasan dari hasil uji coba. Nilai F1 akan diberi warna bergradasi dari putih sampai merah untuk mempermudah visualisasi. Semakin merah warnanya, maka semakin bagus nilainya.

5.4 Skenario Uji Coba

Sebelum melakukan uji coba, perlu ditentukan skenario yang akan digunakan dalam uji coba. Melalui skenario ini, dapat diketahui apakah metode yang diuji sudah berjalan dengan benar dan memiliki performa yang baik sesuai dengan kondisi yang ditentukan. Terdapat tujuh skenario dalam uji coba metode yang terdapat pada Tabel 5.3.

Tabel 5.3 Skenario uji coba

Skenario	Metode	Ukuran Blok	Faktor kontrol sensitivitas
1	*Exact*	8	Sesuai *paper* referensi
2	*Exact*	8	Citra RGB
3	*Exact*	8	Citra RGB dan morfologi opening
4	*Robust*	8	*threshold T, threshold J*
5	*Robust*	8	*threshold T, threshold J, threshold K*
6	*Robust*	16	*threshold T, threshold J*
7	*Robust*	16	*threshold T, threshold J, threshold K*

Tiga skenario pada metode *exact match* dan empat skenario pada metode *robust match*. Setiap skenario dibedakan berdasarkan metode, variabel, atau fitur dengan menggunakan sembilan buah citra yang telah dilakukan serangan *copy-move*. Fitur pendukung merupakan metode yang ada pada *paper* referensi, maupun pengembangan penulis. Pada masing – masing skenario, ditampilkan beberapa contoh citra asli, citra *copy-move*, citra hasil biner, dan citra hasil RGB.

5.4.1 Skenario 1 Metode *Exact Match*

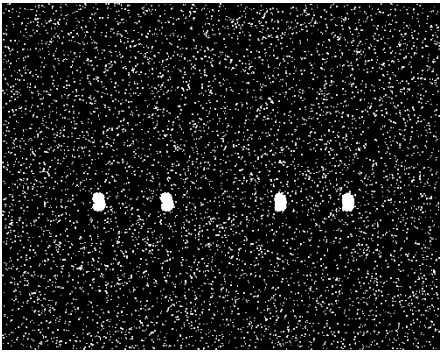
Skenario 1 adalah uji coba metode *exact match* citra *grayscale* dengan menggunakan ukuran blok 8x8 dan dilakukan tanpa operasi morfologi *opening*. Skenario ini sesuai dengan *paper* referensi kecuali pada bagian penyusunan blok, yaitu dengan menggunakan nilai hash.

Tabel 5.4 Hasil Uji Coba Skenario 1

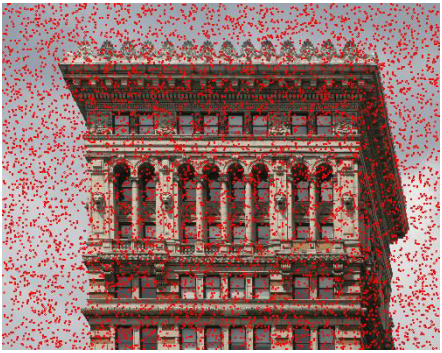
Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	13%	64%	22%	15
clean walls.png	94%	92%	93%	12
giraffe.png	80%	87%	83%	7
knight moves.png	12%	54%	20%	29
malawi.png	100%	45%	62%	192
scotland.png	51%	92%	65%	164
statue.png	7%	94%	12%	107
threehundred.png	30%	80%	43%	73
tree.png	63%	54%	58%	10
Rata-rata	50%	73%	51%	68

Gambar 5.19 *statue.png* asli

Gambar 5.20 *copy-move statue.png*



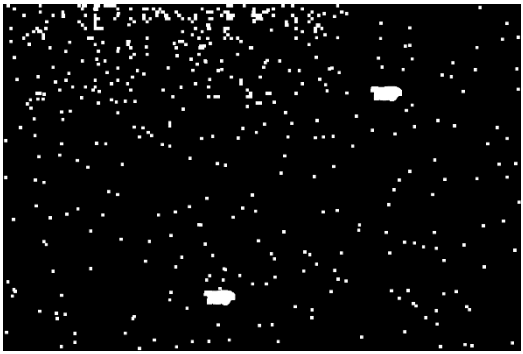
Gambar 5.21 biner *statue.png* hasil skenario 1



Gambar 5.22 RGB *statue.png* hasil skenario 1

Gambar 5.23 *cattle.png* asli

Gambar 5.24 *copy-move cattle.png*



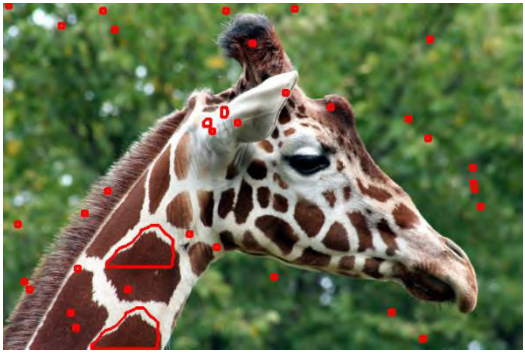
Gambar 5.25 biner *cattle.png* skenario 1

Gambar 5.26 RGB *cattle.png* skenario 1

Gambar 5.27 *giraffe.png* asli

Gambar 5.28 *copy-move giraffe.png*

Gambar 5.29 biner *giraffe.png* hasil skenario 1



Gambar 5.30 RGB *giraffe.png* hasil skenario 1

Tabel 5.4 menunjukkan bahwa skenario 1 menghasilkan berbagai macam nilai dengan kualitas berbeda-beda. Citra *statue.png* mendapatkan *precision* yang paling rendah karena citra *grayscale* dengan ukuran 8x8, algoritma mendeteksi banyak objek yang sama pada objek bukan hasil *copy-move*. Secara umum, skenario ini sudah dapat mengekstraksi objek *copy-move* tetapi masih memiliki tingkat kesalahan deteksi yang besar.

5.4.2 Skenario 2 Metode *Exact Match*

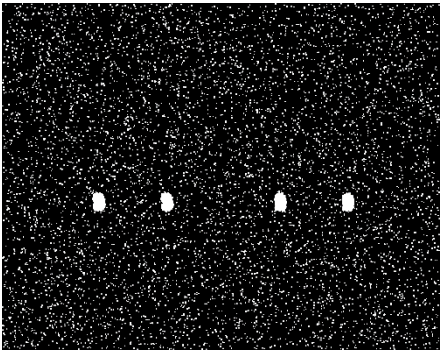
Skenario 2 adalah uji coba metode *exact match* citra RGB dengan menggunakan ukuran blok 8x8 dan dilakukan tanpa operasi morfologi *opening*. Faktor kontrol sensitivitas algoritma disamakan dengan skenario 1 untuk dapat dibandingkan hasil citra *grayscale* dan RGB.

Tabel 5. 5 Hasil Uji Coba Skenario 2

Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	24%	64%	35%	18
clean walls.png	96%	92%	94%	17
giraffe.png	76%	86%	81%	7
knight moves.png	12%	52%	20%	37
malawi.png	100%	43%	60%	138
scotland.png	51%	91%	65%	284
statue.png	6%	94%	12%	115
threehundred.png	31%	79%	44%	88
tree.png	58%	54%	56%	12
Rata-rata	50%	73%	52%	80

Gambar 5.31 *statue.png* asli

Gambar 5.32 *copy-move statue.png*

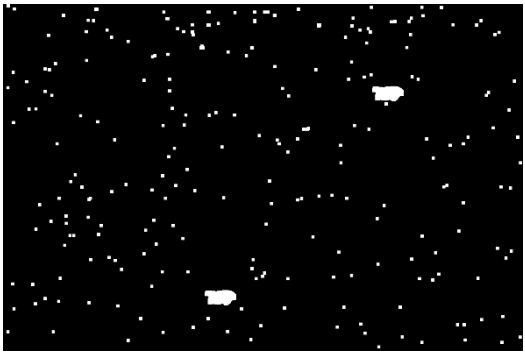


Gambar 5.33 biner *statue.png* hasil skenario 2

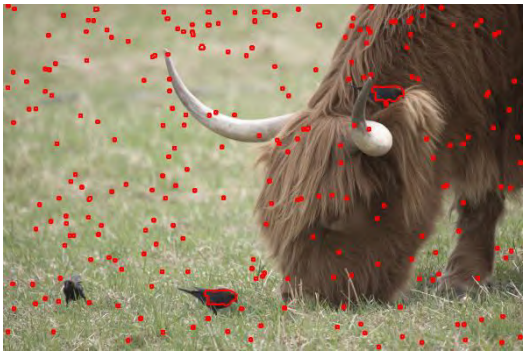
Gambar 5.34 RGB *statue.png* hasil skenario 2

Gambar 5.35 *cattle.png* asli

Gambar 5.36 *copy-move cattle.png*



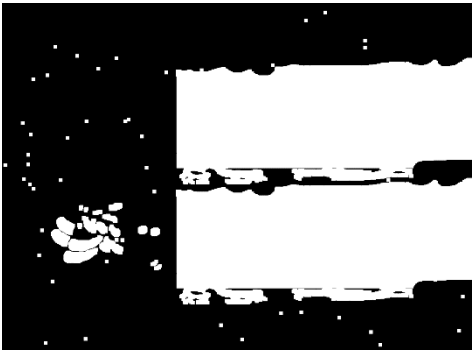
Gambar 5.37 biner *cattle.png* hasil skenario 2



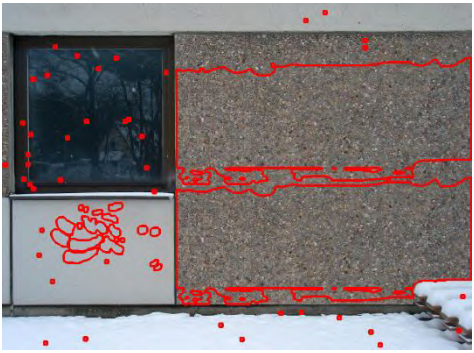
Gambar 5.38 RGB *cattle.png* hasil skenario 2

Gambar 5.39 *clean walls.png* asli

Gambar 5.40 ***copy-move clean walls.png***



Gambar 5.41 biner ***clean walls.png*** **hasil skenario 2**



Gambar 5.42 RGB ***clean walls.png*** **hasil skenario 2**

Tabel 5. 5 menunjukkan penilaian pada skenario 2. Apabila dibandingkan dengan skenario 1, performa kedua skenario tidak jauh berbeda. Nilai rata-rata *precision* dan *recall* dari kedua skenario sama. Kedua skenario memiliki selisih F1 1% di mana skenario 2 mengeluarkan hasil yang lebih baik. Perbedaan yang signifikan terdapat pada citra *cattle.png* karena citra RGB memiliki variasi warna yang lebih beragam dibandingkan *grayscale*.

5.4.3 Skenario 3 Metode *Exact Match*

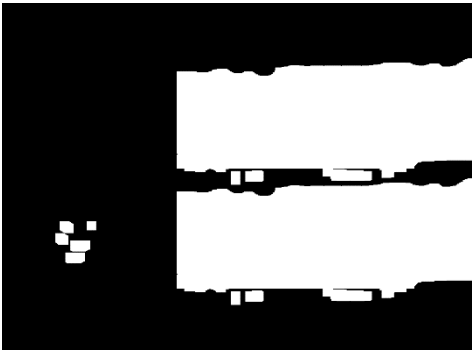
Skenario 3 adalah uji coba metode *exact match* citra RGB dengan menggunakan ukuran blok 8x8 dan dilakukan dengan operasi morfologi *opening*. Skenario 3 dilakukan untuk melakukan uji coba performa morfologi *opening* untuk mengurangi kesalahan deteksi.

Tabel 5.6 Hasil Uji Coba Skenario 3

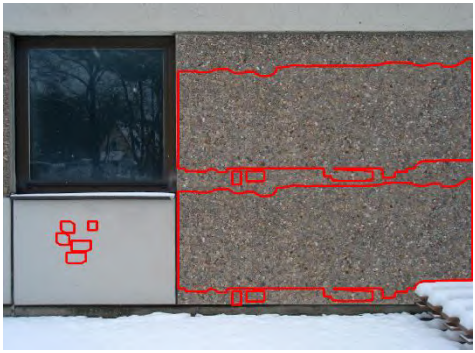
Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	100%	53%	69%	22
clean walls.png	99%	88%	93%	23
giraffe.png	100%	82%	91%	9
knight moves.png	100%	38%	56%	52
malawi.png	100%	35%	51%	227
scotland.png	57%	90%	70%	245
statue.png	100%	89%	94%	105
threehundred.png	100%	74%	85%	105
tree.png	100%	44%	61%	15
Rata-rata	95%	66%	74%	89

Gambar 5.43 *clean walls.png* asli

Gambar 5.44 *copy-move clean walls.png*



Gambar 5.45 biner *clean walls.png* skenario 3

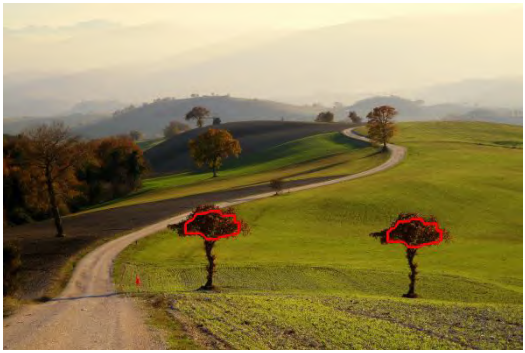


Gambar 5.46 RGB *clean walls.png* skenario 3

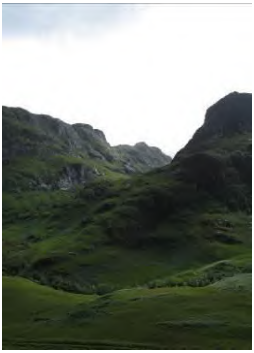
Gambar 5.47 *tree.png* asli

Gambar 5. 48 *tree.png copy-move*

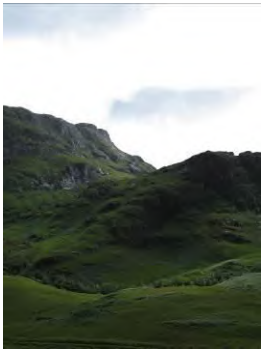
Gambar 5.49 biner *tree.png* skenario 3



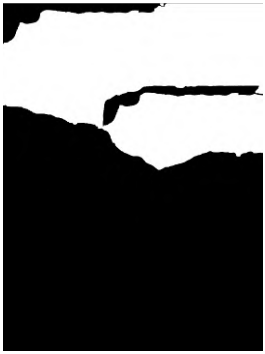
Gambar 5.50 RGB *tree.png* skenario 3



Gambar 5.51 scotland.png asli



Gambar 5.52 scotland.png *copy-move*



Gambar 5.53 biner Scotland skenario 3

Gambar 5.54 RGB Scotland skenario 3

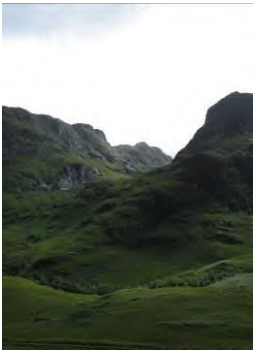
Berdasarkan Tabel 5.6, skenario 3 mengeluarkan hasil yang baik dengan rata-rata kualitas 74%, *precision* 95% dan *recall* 65%. Ini menunjukkan bahwa kecilnya kesalahan deteksi. Morfologi *opening* dapat menghilangkan kesalahan deteksi yang ada pada skenario 2. Citra *clean walls.png* menghasilkan sedikit penurunan kualitas disebabkan objek *copy-move* tertutup oleh morfologi *opening* karena ukuran objek yang kecil, meskipun kesalahan deteksi sudah berkurang.

5.4.4 Skenario 4 Metode *Robust Match*

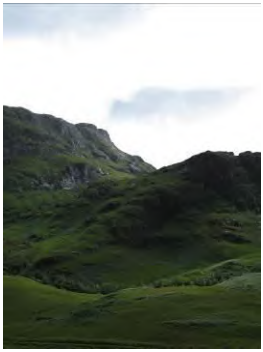
Skenario 4 adalah uji coba metode r*obust match* dengan menggunakan ukuran blok 8x8. Pengujian ini menggunakan *threshold T,* yaitu banyaknya pasangan posisi pada list posisi yang ditemui hasil pencarian dengan nilai 400. Selain itu ditetapkan juga *threshold J,* yaitu jarak antar blok yang sama dengan nilai minimal 50 piksel.

Tabel 5.7 Hasil Uji Coba Skenario 4

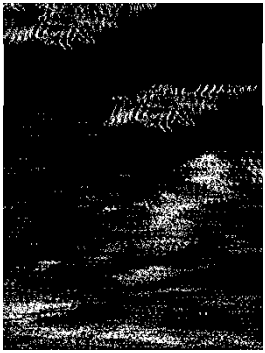
Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	100%	63%	77%	112
clean walls.png	100%	95%	97%	90
giraffe.png	100%	93%	96%	43
knight moves.png	6%	43%	11%	213
malawi.png	80%	64%	71%	758
scotland.png	23%	7%	11%	683
statue.png	12%	96%	22%	677
threehundred.png	100%	88%	94%	411
tree.png	100%	62%	77%	71
Rata-rata	69%	68%	62%	340



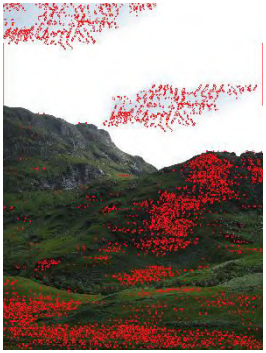
Gambar 5.55 scotland.png asli



Gambar 5.56 scotland.png *copy-move*



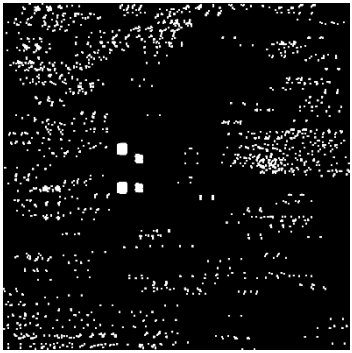
Gambar 5.57 biner Scotland.png skenario 4



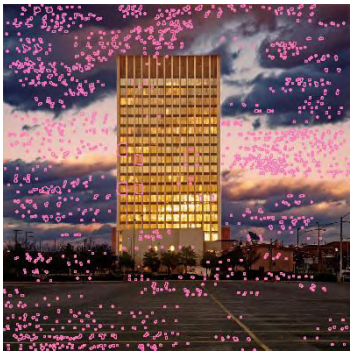
Gambar 5.58 RGB Scotland.png skenario 4

Gambar 5.59 *knight moves.png* asli

Gambar 5.60 knight moves *copy-move*



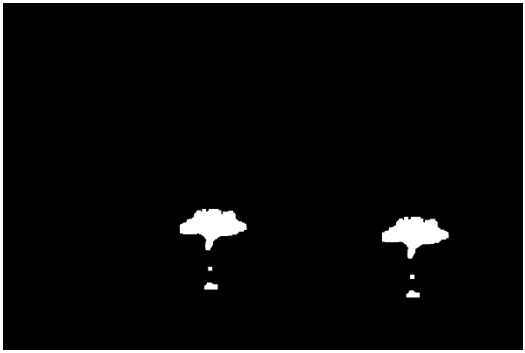
Gambar 5.61 biner *knight moves.png* skenario 4



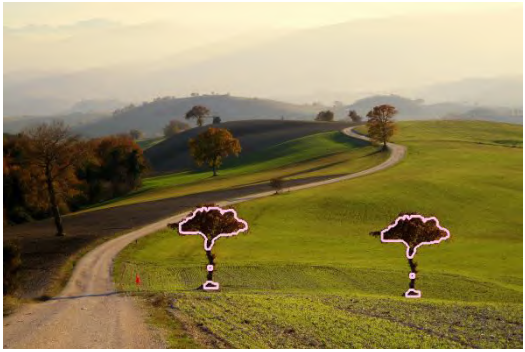
Gambar 5.62 RGB knight moves skenario 4

Gambar 5.63 *tree.png* asli

Gambar 5.64 *tree.png copy-move*



Gambar 5.65 biner *tree.png* skenario 4



Gambar 5.66 RGB *tree.png* skenario 4

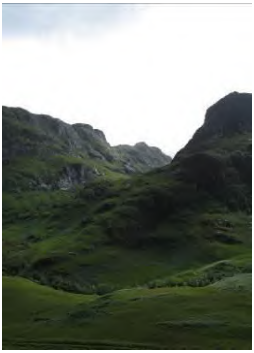
Citra Scotland.png menghasilkan F1 yang jelek dikarenakan terdapat banyak kesalahan deteksi dan objek *copy-move* awan tidak terekstrak dengan sempurna. Citra *knight moves.png* memiliki 6 objek yang dilakukan *copy-move*, tetapi hanya 2 objek yang terdeteksi dan terdapat banyak kesalahan deteksi. Berbeda dengan citra *tree.png* di mana menghasilkan nilai yang lebih baik dibandingkan skenario sebelumnya karena algoritma dapat mengekstrak objek batang pohon yang tidak dapat diekstrak oleh algoritma *exact match*.

5.4.5 Skenario 5 metode *Robust Match*

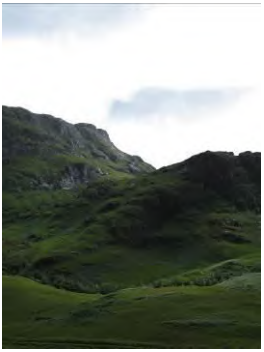
Skenario 5 adalah uji coba metode *robust match* dengan menggunakan ukuran blok 8x8. Pengujian ini menggunakan *threshold T* dengan nilai 400. Ditetapkan juga *threshold J* jarak antar blok yang sama dengan nilai minimal 50 piksel. Selain itu ditetapkan juga *threshold K* jumlah maksimal objek masing-masing *shift-vector* dengan nilai 30.

Tabel 5.8 Hasil Uji Coba Skenario 5

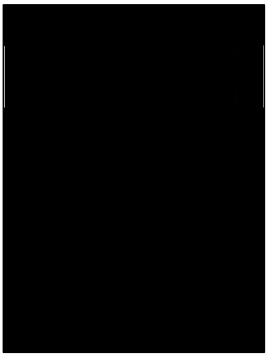
Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	100%	63%	77%	100
clean walls.png	100%	95%	97%	73
giraffe.png	100%	93%	96%	39
knight moves.png	99%	43%	60%	192
malawi.png	100%	15%	26%	656
scotland.png	33%	0%	0%	657
statue.png	0%	0%	0%	517
threehundred.png	100%	88%	94%	351
tree.png	100%	62%	77%	64
Rata-rata	81%	51%	59%	294



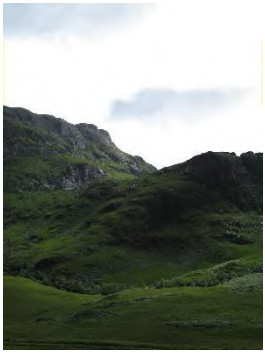
Gambar 5.67 scotland.png asli



Gambar 5.68 scotland.png *copy-move*



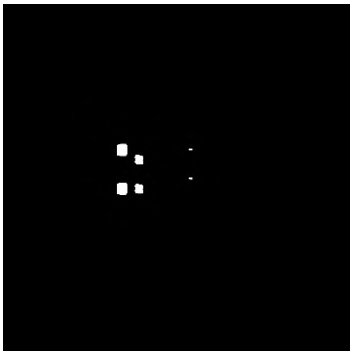
Gambar 5.69 biner Scotland.png skenario 5



Gambar 5.70 RGB Scotland.png skenario 5

Gambar 5.71 *knight moves.png* asli

Gambar 5.72 knight moves *copy-move*



Gambar 5.73 biner *knight moves.png* skenario 5

Gambar 5.74 RGB knight moves skenario 5

Gambar 5.75 *statue.png* asli

Gambar 5.76 *statue.png copy-move*

Gambar 5.77 biner *statue.png* hasil skenario 5

Gambar 5.78 RGB *statue.png* hasil skenario 5

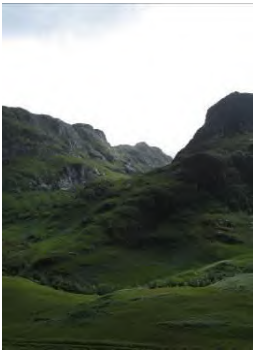
Citra Scotland.png dan *statue.png* menghasilkan nilai F1 0% karena ketatnya *threshold* algoritma. Citra tersebut tidak memiliki *shift-vector counter* yang memiliki objek di bawah *threhold K*. Hal ini berkebalikan dengan citra *knight moves.png* yang memiliki hasil yang lebih baik karena berkurangnya kesalahan deteksi dan objek yang dapat dideteksi lebih banyak dengan menggunakan *threshold K*. Sedangkan beberapa citra lain tidak menunjukkan perubahan yang signifikan.

5.4.6 Skenario 6 metode *Robust Match*

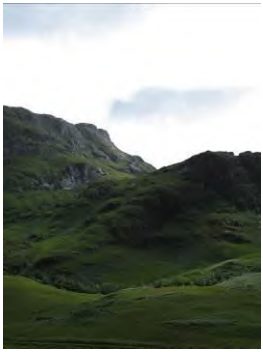
Skenario 6 adalah uji coba metode *robust match* dengan menggunakan ukuran blok 16x16. Pengujian ini menggunakan *threshold T* dengan nilai 400. Selain itu ditetapkan juga *threshold J*, yaitu jarak antar blok dengan nilai 50 piksel.

Tabel 5.9 Hasil Uji Coba Skenario 6

Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	100%	68%	81%	630
clean walls.png	100%	94%	97%	524
giraffe.png	100%	90%	94%	420
knight moves.png	100%	25%	40%	1,319
malawi.png	84%	87%	85%	3,349
scotland.png	96%	17%	29%	2,181
statue.png	81%	93%	87%	3,284
threehundred.png	100%	86%	92%	2,294
tree.png	100%	54%	70%	535
Rata-rata	96%	68%	75%	1,615



Gambar 5.79 scotland.png asli



Gambar 5.80 scotland.png *copy-move*

Gambar 5.81 biner Scotland.png skenario 6

Gambar 5.82 RGB Scotland.png skenario 6

Gambar 5.83 *statue.png* asli

Gambar 5.84 *copy-move statue.png*

Gambar 5.85 biner *statue.png* hasil skenario 6

Gambar 5.86 RGB *statue.png* hasil skenario 6

Berdasarkan Tabel 5.9 skenario 6 mengeluarkan hasil yang cukup baik dengan rata-rata 75%, *precision* 96%, dan *recall* 68%. Dengan ukuran blok yang lebih besar, beberapa citra (*cattle.png*, *statue.png*, Malawi.png, Scotland.png) menghasilkan nilai F1 yang lebih baik dikarenakan berkurangnya kesalahan deteksi atau meminimalisasi terkenanya objek *copy-move* pada *threshold T* dan *threshold J*. Sedangkan citra *giraffe.png*, *threehundred.png*, *knight moves.png*, dan *tree.png* menghasilkan nilai F1 yang sedikit lebih jelek dibandingkan dengan menggunakan ukuran blok

8x8 dikarenakan kecilnya objek sehingga tidak terdeteksi dengan ukuran blok yang besar.

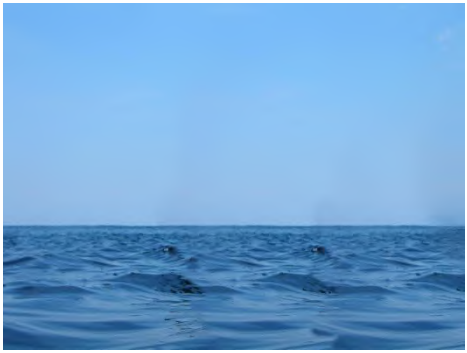
5.4.7 Skenario 7 metode *Robust Match*

Skenario 7 adalah uji coba metode *robust match* dengan menggunakan ukuran blok 16x16. Pengujian ini menggunakan *threshold T* banyaknya pasangan posisi pada list posisi yang ditemui hasil pencarian dengan nilai 400. Ditetapkan *threshold J* jarak antar blok yang sama dengan nilai minimal 50 piksel. Selain itu ditetapkan juga *threshold K* jumlah objek dalam satu *shift-vector* dengan nilai 30.

Tabel 5.10 Hasil Uji Coba Skenario 7

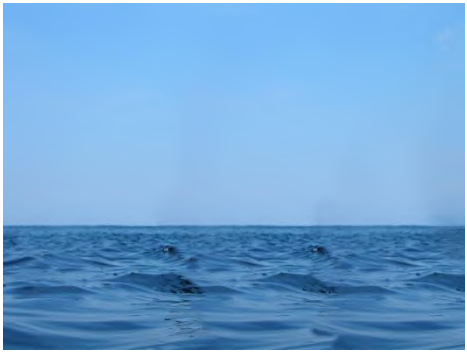
Nama Citra	*Precision*	*Recall*	F1	*Running Time* (detik)
cattle.png	100%	68%	81%	637
clean walls.png	100%	94%	97%	694
giraffe.png	100%	90%	94%	228
knight moves.png	100%	25%	40%	1,619
malawi.png	0%	0%	0%	3,791
scotland.png	95%	15%	26%	2,750
statue.png	81%	93%	87%	3,366
threehundred.png	100%	86%	92%	2,328
tree.png	100%	54%	70%	444
Rata-rata	86%	58%	65%	1,762

Gambar 5.87 malawi.png asli



Gambar 5.88 malawi.png *copy-move*

Gambar 5.89 biner *malawi.png* hasil skenario 7



Gambar 5.90 RGB Malawi.png hasil skenario 7

Berdasarkan Tabel 5.10 skenario 7 mengeluarkan hasil dengan kualitas rata-rata 65%, precision 86%, dan recall 58%. Secara umum, nilai precision dan recall tidak banyak berubah dibandingkan dengan skenario 6. Hal yang mencolok adalah pada citra *malawi.png* yang mendapatkan kualitas 0% dengan visualisasi pada Gambar 5.89. Sisanya skenario ini tidak lebih baik dari skenario 6.

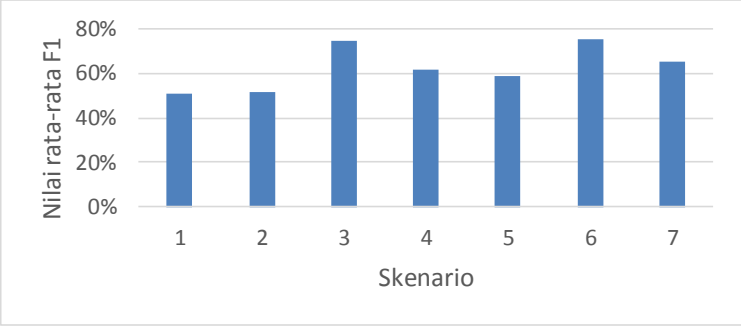
Apabila diperhatikan lebih dalam, citra *malawi.png* akan menghasilkan banyak blok dengan nilai hash yang sama. Terlalu banyaknya blok yang sama dapat menyebabnkan *shift-vector counter* menghasilkan jumlah objek lebih dari nilai *threshold*, yaitu 30. Sehingga isi dari *shift-vector counter* tidak ada yang dikeluarkan.

5.5 Analisa Hasil Uji Coba

Gambar 5.91 menampilkan rata-rata kualitas F1 pada semua skenario uji coba. Dapat dilihat bahwa skenario 3 dan skenario 6 memiliki nilai yang paling tinggi yaitu 74% dan 75%. Skenario 3 menggunakan metode *exact match* dengan ukuran blok 8 dan dengan tambahan operasi morfologi *opening*. Sedangkan skenario 6 menggunakan metode *robust match* dengan ukuran blok 16x16, *threshold J* yang bernilai 50 piksel

jarak antara 2 blok dan *threshold T* 400 piksel sebagai minimal besar objek *copy-move* dalam satu *shift-vector*.

Sedangkan skenario 1 dan 2 menghasilkan kualitas terburuk, di mana kedua skenario adalah metode *exact match*. Minimnya kualitas skenario 1 dan 2 disebabkan oleh banyaknya kesalahan deteksi atau rendahnya *precision* dari kedua skenario.

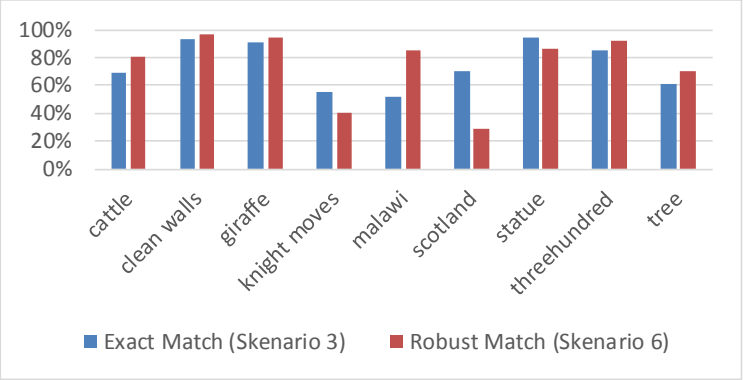


Gambar 5.91 Nilai Rata-rata F1 pada setiap Skenario

Kedua skenario dapat menyelesaikan semua citra yang telah dilakukan serangan *copy-move*. Akan tetapi hasil dari masing-masing citra berbeda-beda. Dari Gambar 5.92 diketahui bahwa hasil skenario 3 lebih merata. Skenario 3 dapat menghasilkan keluaran lebih dari 50% pada semua citra. Sedangkan pada skenario 6 sebagian besar citra memperoleh hasil yang lebih baik dibandingkan skenario 3 kecuali pada citra *knight moves.png* dan s*cotland.png* yang memiliki hasil jauh di bawah.

Pada citra *knight moves.png* dan *Scotland.png* objek atau pasangan blok yang tidak ekstrak disebabkan oleh salahnya deteksi objek yang di *copy-move*. Karena kedua metode hanya melakukan deteksi pada 2 blok yang sama dan akan kesulitan untuk melakukan pengecekan terhadap banyak blok yang sama. Apabila terdapat tiga blok A B C yang memiliki nilai hash yang sama, kedua metode akan melakukan pengecekan A dan B, B

dan C. Sehingga apabila ternyata blok yang dilakukan *copy-move* adalah A dan C, blok tersebut tidak akan terdeteksi.

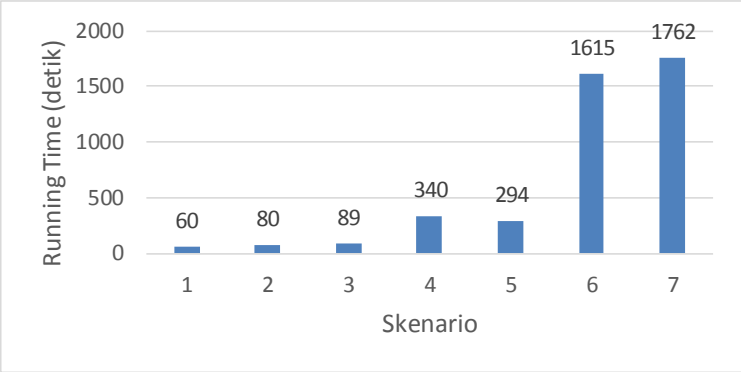


Gambar 5.92. Performa Terbaik Setiap Metode

Kesalahan deteksi biasanya terjadi apabila citra memiliki *background* dengan perbedaan nilai pikselnya sedikit. Beberapa contoh citra yang ada pada dataset adalah *malawi.png*, *scotland.png*, *statue.png*. Citra tersebut memiliki *precision* yang kecil pada skenario 6. Sedangkan *Scotland* memiliki nilai yang sangat rendah di kedua metode. Hal ini disebabkan karena citra *Scotland* memiliki piksel berwarna putih dengan intensitas yang tinggi sehingga sulit membedakan antar blok. Respons dari kedua metode berbeda karena adanya penerapan *threshold* pada metode 6. Pada skenario 3 *scotland.png* memiliki nilai *precision* yang rendah dan *recall* yang tinggi. Sedangkan pada skenario 6 memiliki *precision* yang tinggi dan *recall* yang rendah.

Pada citra *knight moves.png* juga terdapat kasus yang sama. Akan tetapi kesalahan deteksi bukan pada *background*, melainkan banyak objek yang serupa dengan objek serangan *copy-move*. Objek-objek yang salah deteksi tidak diekstrak karena adanya *threshold*. Sehingga tidak menimbulkan *FP*

tetapi mengurangi *TP*. Oleh karena itu *knight moves.png* memiliki nilai *recall* yang rendah.



Gambar 5.93 Rata-rata *Running Time* Setiap Skenario

Dilihat dari sisi waktu berjalannya metode *exact match* pada skenario 1,2, dan 3 berjalan jauh lebih cepat dibandingkan metode *robust match* pada skenario 4,5,6,7, dan 8. Hal ini terjadi karena pada *robust match* dilakukan perhitungan DCT yang memiliki kompleksitas tinggi. Dilihat dari skenario 4 dan 5 dengan ukuran blok 8, dan skenario 6 dan 7 dengan ukuran blok 16, ro*bust match* dengan ukuran blok yang lebih besar memakan waktu yang besar pula, karena semakin besar ukuran blok, semakin banyak perhitungan DCT yang dilakukan. Ukuran blok 16x16 membutuhkan perhitungan DCT empat kali pada setiap blok. Sedangkan ukuran blok 8x8 hanya melakukan perhitungan DCT satu kali setiap blok.

Apabila dilakukan perbandingan antara waktu dan hasil yang diberikan, metode *exact-match* sudah cukup dapat mendeteksi serangan *copy-move* dengan visual. Akan tetapi apabila ingin mendapatkan kualitas yang lebih baik, metode *robust match* lebih cocok dengan risiko membutuhkan waktu yang lebih lama. Metode *exact match* memiliki ciri khas

tingginya nilai *precision* sedangkan *robust match* memiliki ciri khas tingginya nilai *recall*.

BAB VI
KESIMPULAN DAN SARAN

Bab VI ini membahas tentang kesimpulan yang didasari oleh hasil uji coba pada bab sebelumnya. Kesimpulan tersebut nantinya menjawab rumusan masalah yang telah ada pada pendahuluan. Selain itu, juga terdapat saran sebagai acuan untuk mengembangkan topik Tugas Akhir ini lebih lanjut di masa depan.

6.1. Kesimpulan

Dari hasil uji coba yang telah dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Secara umum metode *robust match* menghasilkan kualitas yang lebih baik dibandingkan *exact match*. Akan tetapi metode *robust match* membutuhkan waktu yang lebih lama.
2. Metode *robust match* dengan ukuran blok 16x16 piksel dengan tambahan *threshold* pada jarak blok sebesar 50 piksel dan *threshold* pada banyaknya blok yang terdeteksi dengan jarak yang sama sejumlah minimal 400 pasang blok merupakan faktor kontrol sensitivitas metode terbaik dengan hasil sampai dengan 97%.
3. Faktor kontrol sensitivitas terbaik dengan menggunakan *exact match* adalah ukuran blok 8x8 piksel dan dilakukan morfologi *opening* sebesar 20x20 piksel pada tahap akhir.
4. Metode *robust match* memiliki kekurangan dalam menghadapi banyaknya blok dengan nilai yang sama karena terbatasi oleh *threshold* yang telah ditetapkan.
5. Metode *exact match* memiliki kekurangan dalam toleransi kesamaan blok.

6.2. Saran

Saran yang diberikan untuk pengembangan metode ini adalah:

1. Melakukan perhitungan DCT dan kuantisasi pada semua bagian citra terlebih dahulu, kemudian mengambil blok-blok yang berkaitan untuk disusun dalam mendapatkan blok dengan ukuran lebih besar dari 8x8 sehingga waktu pemrosesan dapat berjalan lebih cepat.
2. Penggabungan metode *exact match* pada *robust match* untuk meminimalisasi kesalahan deteksi.
3. Analisa nilai setiap *threshold* pada metode *robust match*.
4. Melakukan pengembangan untuk mengatasi banyak blok dengan nilai yang sama pada metode *robust match*.
5. Pada *paper* [5] dengan menggunakan DCT mendapatkan kualitas rata-rata 84%, sehingga Tugas Akhir ini masih dapat dikembangkan lagi dengan menggunakan metode-metode tambahan seperti melakukan perbandingan dengan menggunakan *nearest neightbor* atau yang lainnya.

Daftar Pustaka

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

[2] P. Albert J. Marcella, Cyber Forensics - A Field Manual of Collectiong, Examining, and Preserving Evidence of Computer Crimes, CRC Press Book, 2002.

[3] J. A. Redi, W. Taktak dan J.-L. Dugelay, "Digital Image Forensics: a booklet for beginners".

[4] D. Putra, Pengolahan Citra Digital, Yogyakarta: Penerbit Andi, 2010.

[5] V. Christlein, C. Riess, J. Jordan, C. Riess dan E. Angelopoulou, "An Evaluation of Popular Copy-Move Forgery Detection Approaches," *IEEE Transactions on Information Forensics and Security,* vol. 7, pp. 1841-1854, 2012.

[6] M. Lutz, Learning Python, Sebastopol: O'Reilly Media, 2013.

[7] G. Bradski dan A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, Sebastapol: O'Reilly Media, 2008.

[8] I. Idris, NumPy Beginner's Guide (Second Edition), Brimingham: Packt Publishing, 2013.

[9] H. Delfs dan H. Knebl, Introduction to Cryptography: Principles and Application Second Edition, Heidelberg: Springer, 2007.

[10] (FAU) Friedrich-Alexander-Universität Erlangen-Nürnberg, "Image Manipulation Dataset," [Online]. Available: https://www5.cs.fau.de/research/data/image-manipulation/. [Diakses 10 June 2016].

BIODATA PENULIS

Sa'id Al Musayyab, lahir di Jakarta pada tanggal 5 Januari 1994. Penuliss menempuh pendidikan mulai dari SDIT Al - Furqon Jakarta (2000-2006), SMP Al Hikmah Surabaya (2006-2009), SMA Al Hikmah Surabaya (2009-2012), dan sekarang sedang menjalani pendidikan S1 Teknik Informatika di ITS. Penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer (HMTC). Di antaranya adalah menjadi staf departemen hubungan luar Himpunan Mahasiswa Teknik Computer ITS 2013-2014 dan staf ahli hubungan luar di Himpunan Mahasiswa Teknik Computer ITS 2014-2015. Penulis juga aktif dalam kegiatan kepanitiaan Schematics. Di antaranya penulis pernah menjadi staf humas Schematics 2013 dan 2014. Penulis mengambil bidang minat Komputasi Berbasis Jaringan (KBJ). Komunikasi dengan penulis dapat melalui email: **saidmusayyab@gmail.com**.