PERANCANGAN APLIKASI PENGOLAHAN CITRA DIGITAL UNTUK MENDETEKSI HAMA DAN PENYAKIT PADA TANAMAN JAGUNG

SKRIPSI

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun Oleh :

Faiza Alif Fakhrina

NIM. 0710683018

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

PROGRAM STUDI TEKNIK INFORMATIKA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2013

LEMBAR PERSETUJUAN

PERANCANGAN APLIKASI PENGOLAHAN CITRA DIGITAL UNTUK MENDETEKSI HAMA DAN PENYAKIT PADA TANAMAN JAGUNG

SKRIPSI

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun Oleh :

Faiza Alif Fakhrina

NIM. 0710683018

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
Suprapto, S.T, M.T.	**Ir. Heru Nurwasito, M.Kom**
NIP. 197107271996031001	**NIP. 196504021990021001**

LEMBAR PENGESAHAN

PERANCANGAN APLIKASI PENGOLAHAN CITRA DIGITAL UNTUK MENDETEKSI HAMA DAN PENYAKIT PADA TANAMAN JAGUNG

SKRIPSI

KONSENTRASI REKAYASA PERANGKAT LUNAK

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :

Faiza Alif Fakhrina

NIM. 0710683018

Skripsi ini telah diuji dan dinyatakan lulus pada

tanggal 11 Januari 2013

Penguji I	Penguji II
Budi Darma Setiawan, S. Kom., M.Cs.	**Satrio Agung W., S,Kom., M.Kom.**
NIK. 841015 06 1 1 0090	**NIK. 86052106110114**

Penguji III

Ahmad Afif Supianto, S. Si., M. Kom.

Mengetahui
Ketua Program Studi Teknik Informatika

Drs. Marji, M.T.

NIP. 19670801 199203 1 001

PERNYATAAN
ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 17 Desember 2012

Mahasiswa,

Faiza Alif Fakhrina

NIM 0710683018

KATA PENGANTAR

Puji syukur kehadirat Tuhan SWT pemeliharaan seluruh alam raya, sebagai pencipta atas segala kehidupan yang kita lihat, kita dengar dan kita rasa yang telah melimpahkan rahmat dan hidayahNya, sehingga penulis dapat menyelesaikan skripsi yang berjudul "Sistem Pengolahan Citra Digital Untuk Mendeteksi Hama Dan Penyakit Tanaman Jagung."

Saya menyadari, skripsi yang saya tulis itu bukan merupakan suatu yang instant. Itu buah dari suatu proses yang relatif panjang, menyita segenap tenaga dan fikiran. Skripsi ini merupakan tugas akhir yang diajukan untuk menempuh ujian sarjana dengan gelar Sarjana Komputer (S.Kom) pada Program Teknik Informasi dan Ilmu Komputer Universitas Brawijaya Malang.

Terima kasih dan rasa syukur yang dalam, saya sampaikan kepada kedua orang tua saya (Bapak dan Mama) yang telah membesarkan saya dan mendidik saya dengan kesabaran, kasih dan doa. Sungguh, kasih kalian adalah kepanjangan tangan Tuhan untuk hidup saya. Oleh karena itu, dalam kesempatan ini, penulis juga ingin mengucapkan terima kasih dengan hati yang tulus kepada:

1. Ayahanda Muhammad Fakhry, Ibunda Nining Suhermin, Adik Syifauts Tsany Fakhrina dan seluruh keluarga besar atas segala nasehat, kasih sayang, perhatian dan kesabarannya di dalam membesarkan dan mendidik penulis, serta yang senantiasa tiada henti – hentinya memberikan doa dan semangat demi terselesaikannya skripsi ini.
2. Bapak Ir. Sutrisno, M.T, Bapak Ir. Heru Nurwasito, M.Kom, Bapak Himawat Aryadita, S.T, M.Sc, dan Bapak Eddy Santoso, S.Kom selaku Ketua, Wakil Ketua 1, Wakil Ketua 2 dan Wakil Ketua 3 Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya.
3. Bapak Drs. Marji, M.T dan Bapak Issa Arwani, S.Kom, M.Sc selaku Ketua dan Sekretaris Program Studi Teknik Informatika Universitas Brawijaya.

Bapak Suprapto, ST., MT., dan Bapak Ir. Heru Nurwasito, M.Kom selaku dosen pembimbing skripsi yang telah dengan sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.

4. Bapak Himawat Aryadita, S.T, M.Sc selaku dosen penasehat akademik yang selalu memberikan nasehat kepada penulis selama menempuh masa studi.
5. Seluruh Dosen Teknik Informatika Universitas Brawijaya atas kesediaan membagi ilmunya kepada penulis.
6. Seluruh Civitas Akademika Teknik Informatika Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan selama penulis menempuh studi di Teknik Informatika Universitas Brawijaya dan selama penyelesaian skripsi ini.
7. Unnyku Verbyan Arizona yang telah bersedia menemani serta memberikan dukungan dan semangat dalam menyelesaikan tugas akhir ini.
8. Sahabatku Noviana Putri Pradnyawati, Arif Mudjahidah, Ratih Kartika Dewi, Adam Hendra Brata, Sativandi Putra, Cantika Previana yang telah banyak memberikan bantuan dan dukungan dalam menyelesaikan tugas akhir ini.
9. Teman-teman Legendary TPL'07 yang telah memberikan semangat dalam menyelesaikan tugas akhir.
10. Mbak Ririn Bintari Putri, Dwi Apriliani B. aka Bataq, Ika Nurin R. yang selama ini telah membantu memberikan semangat dan menghibur untuk meneruskan perjuangan tugas akhir hingga selesai.

Akhir kata, karena tiada gading yang tak retak, peneliti menyadari bahwa skripsi ini masih memiliki banyak kekurangan. Untuk itu, penulis mengharapkan kritik dan saran dari pembaca dan kiranya skripsi ini dapat memberi manfaat bagi penulis maupun pihak lain yang menggunakannya.

Malang, 21 Desember 2012

Penulis

ABSTRAK

Faiza Alif Fakhrina. 2013. : Perancangan Aplikasi Pengolahan Citra Digital Untuk Mendeteksi Hama Dan Penyakit Pada Tanaman Jagung.
Dosen Pembimbing : Suprapto, ST., MT., dan Ir. Heru Nurwasito, M.Kom.

Gabungan antara prediksi suatu citra hama dengan pemanfaatan kemajuan teknologi informasi dapat dijadikan sebagai media berupa sistem pengolahan citra digital. Sistem pengolahan citra digital merupakan suatu aplikasi yang mengolah citra digital kemudian melakukan prediksi terhadap citra hama tanaman jagung untuk membantu kinerja petani. Aplikasi yang dibuat dalam penelitian ini berupa sistem pengolahan citra digital untuk memprediksi hama dan penyakit jagung menggunakan metode *PrincipalComponentAnalysis* (PCA).

Sistem pengolahan citra digital ini dirancang menggunakan OOAD (*Object Oriented Analysis and Design*) dan diimplementasikan menggunakan bahasa pemrograman Java. Perangkat lunak sistem ini dikembangkan dengan metode *waterfall*. Pengujian sistem ini meliputi pengujian perangkat lunak dan pengujian validitas sistem. Pada pengujian perangkat lunak sistem ini menggunakan metode *black-box testing* dan *white-box testing*. Sedangkan pengujian validitas sistem dilakukan dengan menguji kualitas citra digital dan menguji validasi citra test dengan cropping juga menguji validasi citra test dengan menyelipkan objek pada citra test sebagai input. Hasil pengujian kualitas citra dengan *noise* ini menunjukkan kualitas citra yang diberikan *noise* sebesar 400% maka hasil prediksi citra tetap sesuai dengn data training. Untuk hasil pengujian validasi citra dengan cropping dan menyelipkan objek pada citra test menunjukkan beberapa citra test tidak dapat diprediksi karena terdapat beberapa bagian hama yang hampir mirip dengan bagian tubuh hama lainnya. Hasil pengujian kualitas citra dan validasi citra test ini juga menunjukkan bahwa pose dan bentuk hama data test sangat berpengaruh terhadap hasil prediksi citra.

Kata Kunci: pengolahan citra digital, *Principal Component Analysi, waterfall.*

ABSTRACT

Faiza Alif Fakhrina. 2013. : Design of Digital Image Processing Applications to Detect Pests And Diseases In Corn Plants.
Dosen Pembimbing : Suprapto, ST., MT., dan Ir. Heru Nurwasito, M.Kom.

The combination of a prediction image of pests with the use of advances in information technology can be used as medium of digital image processing system. Digital image processing system is an application that processes the digital image and then make predictions on the image of the corn crop pests to help the performance of farmers. Application made in the study of digital image processing system for predicting corn pests and diseases using *PrincipalComponentAnalysis* (PCA) method.

Digital image processing system is designed to use OOAD (Object Oriented Analysis and Design) and implemented using the Java programming language. The software system was developed with the waterfall method. Tests of this system include software testing and testing the validity of the system. In software testing system using black-box testing and white-box testing. Testing the validity of the system is done by testing digital image quality and test validation test image by cropping the image test also test by inserting objects in test images as input. The results of the noise image quality testing shows the quality of the given image noise at 400% of the predicted results remain appropriate image with less training data. For the validation test results with image cropping and image of the test object on the slip shows some test images can not be predicted because there are some parts of the pests that resemble body parts of other pests. The results of testing the quality of the image and image validation test also showed that the pose and shape of pests test data greatly affect the results of the prediction image.

Kata Kunci: *digital image processing, Principal Component Analysi, waterfall.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Seiring perkembangan teknologi dan informasi yang semakin pesat ini memicu adanya peningkatan di berbagai aspek kehidupan. Tidak hanya di bidang pendidikan, pariwisata, ekonomi hingga pertanian yang mengikuti perkembangan teknologi dan informasi. Dengan adanya peningkatan ini dapat mempermudah manusia dalam melakukan pekerjaannya dengan cepat.

Pertanian mempunyai arti penting dalam kehidupan manusia di dunia selama manusia hidup maka selama itu pula pertanian akan ada. Hal itu disebabkan karena makanan merupakan kebutuhan pokok selain udara dan air yang dibutuhkan setiap harinya. Makanan merupakan hasil dari pertanian yang akan dihasilkan tiap tahunnya untuk memenuhi kebutuhan manusia yang semakin tahun mengalami peningkatan jumlah. Salah satu makanan pokok masyarakat Indonesia adalah beras, tetapi di beberapa daerah terpencil di Indonesia jagung menjadi makanan pokok selain beras.

Kerusakan yang sering terjadi pada umumnya adanya serangan hama dan penyakit pada tanaman jagung ini sehingga menghambat pertumbuhan tanaman jagung secara maksimal yang berdampak pada pemenuhan kebutuhan pangan masyarakat Indonesia. Pada umumnya sebelum tanaman tersebut mengalami kerusakan yang parah dan meluas akan menunjukkan beberapa gejala yang bisa didiagnosis sebelumnya, tetapi para petani sering mengabaikan adanya gejala tersebut. Ini dikarenakan ketidaktahuan petani dan menganggap hal tersebut sudah biasa terjadi sehingga membiarkan hama dan penyakit tersebut berkembang biak dan menyebar luas.

Oleh karena itu, untuk menunjang peningkatan hasil pertanian jagung di Indonesia perlu dibuat sebuah aplikasi komputasi yang memudahkan petani untuk mengenali hama dan penyakit yang terkena tanaman jagung tersebut. Dengan menggunakan citra digital diharapkan para petani bisa mengenali hama dan penyakit tersebut. Yang pada akhirnya dapat mengurangi resiko rusaknya hasil tanaman jagung tersebut.[EDY-09]

Untuk mengolah citra digital hama jagung menggunakan teori ekstraksi fitur yaitu *Principal Component Analysis* seperti yang telah dilakukan sebelumnya oleh Riza

Masitha Wati untuk pengenalan pola senyum pada wajah manusia. Input yang digunakan untuk proses diatas sama-sama menggunakan citra digital (foto) sehingga proses pengenalan yang dilakukan tidak berbeda hanya data masukan yang digunakan untuk prediksi citra adalah foto hama jagung.[WAT-11]

Untuk mengolah data masukan hama dan penyakit tanaman jagung ini menggunakan citra digital sebagai *input* masukan yang kemudian diolah dengan metode ekstraksi fitur yaitu *PrincipalComponentAnalysis* (PCA). Hasil *output* citra tersebut berupa nilai *matrix* bobot yang nantinya akan disamakan dengan nilai bobot citra *training* hama dan jika nilai yang didapat sama maka *output* citra yang muncul adalah citra yang sesuai dengan citra *test* hama.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang dirumuskan di atas, maka rumusan masalah sebagai berikut :

1. Bagaimana menginterpretasikan algoritma yang ada dalam kasus ini untuk mendeteksi hama dan penyakit jagung dalam citra digital.
2. Bagaimana merancang sebuah aplikasi yang dapat mendeteksi hama dan penyakit jagung dalam suatu citra digital.
3. Bagaimana mengetahui performansi ekstraksi fitur yaitu *PrincipalComponentAnalysis*.

1.3 Batasan Masalah

Dengan mengacu pada permasalahan yang telah dirumuskan, maka hal-hal yang berkaitan dengan sistem akan diberi batasan sebagai berikut:

1. Bahasa pemrograman yang digunakan adalah *Java*.
2. Sistem operasi yang digunakan adalah *Windows* 7.
3. *Platform* pengembangan yang digunakan adalah *JavaStandartEdition*.
4. Data masukan yang dapat diolah adalah .jpg .
5. Penyakit dan hama yang dapat dikenali dalam sistem ini hanya masalah yang disebabkan oleh hewan dapat dikenali bentuknya, misal ulat grayak,belalang, tikus, penggerek batang jagung, dll.

6. Teori pengolahan citra dengan metode ekstraksi fitur *PrincipalComponentAnalysis* tidak dibahas secara detail karena penelitian ini terbatas pada materi Rekayasa Perangkat Lunak dari rancangan aplikasi ini.
7. Pembahasan difokuskan pada Rekayasa Perangkat Lunak yang terdiri dari *use case diagram, activity diagram, sequence diagram* serta *testing* (*white box* dan *black box*)dari aplikasi pengolahan citra digital untuk mendeteksi hama dan penyakit tanaman jagung.

1.4 Tujuan

Penyusunan skripsi ini dengan tujuan, antara lain :

1. Membuat rancangan sistem pengolahan citra digital untuk mendeteksi hama dan penyakit tanaman jagung yang dapat digunakan petani untuk membantu diagnosa hama dan penyakit pada tanaman jagung yang ditanam.
2. Aplikasi ini dapat mempermudah, meningkatkan kinerja dan dapat menggantikan kinerja manual yang ada di lingkup petani dan ahli pertanian.

1.5 Manfaat

Manfaat penelitian ini antara lain:

a. Bagi penulis:
   1. Menerapkan ilmu yang telah diperoleh dari Program Studi Teknik Informatika Universitas Brawijaya.
   2. Mendapatkan pemahaman tentang perancangan aplikasi pengolahan citra digital untuk mendeteksi hama dan penyakit tanaman jagung menggunakan Java.

b. Bagi pengguna:

   Mempermudah dalammendiagnosa hama dan penyakit pada tanaman jagung tanpa harus menunggu hasil yang lama.

1.6 Sistematika Pembahasan

Sistematika pembahasan dalam skripsi ini sebagai berikut:

BAB I Pendahuluan

Bab ini berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika pembahasan.

BAB II Dasar Teori

Bab ini berisi tentang tinjauan pustaka dan landasan teori.

BAB III Metodologi

Bab ini membahas metode yang digunakan dalam penulisan yang terdiri dari studi literatur, perancangan perangkat lunak, implementasi perangkat lunak, pengujian dan analisis, serta pengambilan kesimpulan dan saran.

BAB IV Perancangan

Bab ini berisi tentang rencana pelaksanaan, alat, bahan, jalannya perencanaan dan hasil yang diharapkan.

BAB V Implementasi

Bab ini berisi tentang gambaran sistem yang akan dirancang dan deskripsi sistem hasil analisis kebutuhan sistem, perancanga sistem, implementasi, dan analisis pengujian.

BAB VI Pengujian dan Analisis

Memuat hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

BAB VII Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian program, serta saran - saran untuk pengembangan lebih lanjut.

BAB II

KAJIAN PUSTAKAN DAN DASAR TEORI

2.1 Citra Digital

Menurut arti secara harfiah, citra (*image*) adalah gambar pada bidang dua dimensi. Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*continue*) dari intensitas cahaya pada bidang dua dimensi. [HAR-06]

Citra sebagai *output* dari suatu sistem perekaman data dapat bersifat:

- Optik, berupa foto,
- Analog berupa sinyal video, seperti gambar pada monitor televisi,
- Digital yang dapat langsung di simpan pada suatu pita *magnetic*.

Citra digital merupakan suatu fungsi intensitas cahaya f(x,y), dimana harga x dan y merupakan koordinat spasial dan harga fungsi tersebut pada setiap titik (x,y) merupakan tingkat kecemerlangan citra pada titik tersebut. [BAN-10]

Citra digital dinyatakan dengan matriks berukuran N x M (N menyatakan baris atau tinggi, M menyatakan kolom atau lebar).

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix}$$

Keterangan :

N = jumlah baris, $0 \le y \le N-1$

M = jumlah kolom, $0 \le x \le M-1$

L= maksimal warna intensitas (derajat keabuan), $0 \le f(x,y) \le L-1$

Citra digital biasanya berbentuk persegi panjang, secara visualisasi dimensi ukurannya dinyatakan sebagai lebar x tinggi. Ukurannya dinyatakan dalam titik atau piksel (*pixel = picture element*) dan dapat pula dinyatakan dalam satuan panjang (mm atau inci = *inch*).[TAU-09]

Berdasarkan format penyimpanan nilai warnanya, citra terdiri dari atas empat jenis ,yaitu:

- Citra *biner* atau monokrom

Pada citra jenis ini, setiap titik atau piksel hanya bernilai 0 atau 1. Dimana setiap titik membutuhkan media penyimpanan sebesar 1 bit. Gambar 2.1 merupakan contoh citra *biner*.

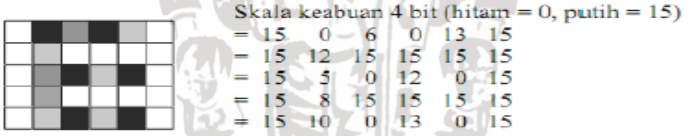
Gambar 2.1 Citra Biner

Sumber:[TAU-09:20]

- Citra skala keabuan

Citra skala keabuan mempunyai kemungkinan warna antara hitam (minimal) dan putih (maksimal). Jumlah maksimum warna sesuai dengan bit penyimpanan yang digunakan. Misal :

Suatu citra dengan skala 4bit, memiliki jumlah kemungkinan warna 2^4 = 16 warna. Gambar 2.2 memperlihatkan citra skala keabuan 4bit.



Gambar 2.2 Citra Skala Keabuan

Sumber:[TAU-09:21]

- Citra warna (*true colors*)

Setiap titik (piksel) pada citra warna mewakili warna yang merupakan kombinasi dari tiga warna dasar yaitu merah, hijau dan biru yang dikenal sebagai citra RGB (*Red*, *Green*, *Blue*). Setiap warna dasar mempunyai intensitas sendiri dengan nilai maksimum 255 (8bit).

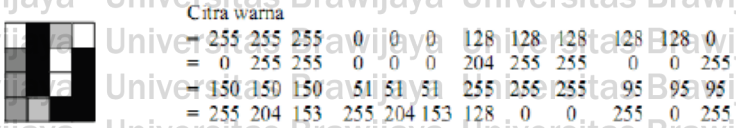
Red = warna minimal putih, warna maksimal merah.

Green = warna minimal putih, warna maksimal hijau.

Blue = warna minimal putih, warna maksimal biru.

Setiap titik pada citra membutuhkan data 3*byte*. Jumlah kemungkinan kombinasi warna untuk citra warna adalah 2^{24} = lebih dari 16 juta warna, disebut *true color* karena

dianggap mencakup semua warna yang ada. Gambar 2.3 memperlihatkan contoh citra warna.

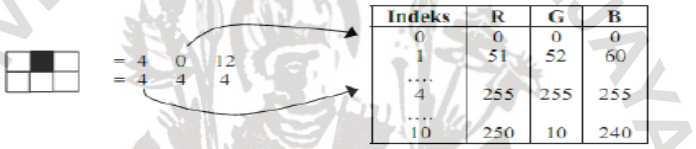


Gambar 2.3 Citra Warna (*True Color*)

Sumber:[TAU-09:21]

➢ Citra warna berindeks

Setiap titik (piksel) pada citra warna berindeks mewakili indeks dari suatu tabel warna yang tersedia (biasanya disebut palet warna). Keuntungan pemakaian palet warna adalah kita dapat dengan cepat memanipulasi warna tanpa harus mengubah informasi pada setiap titik dalam citra. Keuntungan yang lain, penyimpanan lebih kecil. Contoh citra warna berindeks diperlihatkan pada Gambar 2.4.



Gambar 2.4 Citra Warna Berindeks

Sumber:[TAU-09:22]

Terdapat beberapa tipe *file* citra digital yang dikenal saat ini, diantaranya JPEG (*Joint Photographic Experts Group*), BMP (*Bitmap*), GIF (*Graphics Interchange Format*), TIFF (*Tagged Image File Format*), dan PNG (*Portable Network Graphics*). Dalam skripsi ini *file* citra digital yang digunakan adalah citra digital denga tipe *file* JPEG. Citra JPEG merupakan tipe *file* citra digital yang banyak digunakan untuk menyimpan gambar-gambar dengan ukuran lebih kecil. Citra JPEG memiliki beberapa karakteristik, yaitu :

- Memiliki ekstensi .jpg atau .jpeg
- Mampu menayangkan warna dengan kedalaman 24-bit *true color*
- Mengkompresi gambar dengan sifat *lossy*
- Umumnya digunakan untuk menyimpan gambar-gambar hasil foto

2.2 *PrincipalComponentAnalysis* (*PCA*)

Pengertian *Principal Component Analysis* (*PCA*) menurut kutipan buku "konsep pengolahan Citra Digital dan Ekstraksi Fitur oleh Mauridhi Hery Purnomo dan Arif Muntasa dari Turk,1991 merupakan teknik *linear* reduksi menggunakan teori sederhana dari statistik.

Misalkan diketahui suatu citra dengan ukuran h x w, dengan h adalah tinggi citra dan w adalah lebar citra, maka dimensi citra adalah n dengan n = h x w. Jika suatu citra dengan ukuran n, maka jumlah kombinasi linier dari citra sebanyak n merupakan dimensi yang tinggi, dan ini merupakan masalah besar pada komputasi ketika proses pengukuran. Dengan menggunakan PCA sebagai ekstraksi fitur, maka dimensi yang tinggi tersebut dapat dikurangi menjadi dimensi yang rendah. Jumlah dimensi yang dihasilkan oleh PCA tergantung pada jumlah data yang digunakan oleh pelatihan dan jumlah *sampel* pada masing-masing data pelatihan. Misalkan jumlah data pelatihan adalah k dan masing-masing data mempunyai s model, maka jumlah *sampel* keseluruhan adalah m dengan m = k x s. Metode PCA efisien digunakan pada suatu kondisi m << n (jumlah data jauh lebih kecil dibandingkan dengan dimensi citra yang digunakan penelitian). [PUR-10:221]

Sebelum membentuk algoritma PCA, ada beberapa hal yang perlu diketahui pada proses PCA, diantaranya adalah : [PUR-10:221]

a. Setiap data citra harus dibentuk menjadi matrik baris atau matrik kolom. Contoh, misalkan citra yang digunakan pada kasus ini mampunyai ukuran tinggi 165 dan lebar 110, maka dibentuk matrik bari menjadi dimensi 1x165.000. jika jumlah data yang akan digunakan adalah 50 dan masing-masing sampel ada 4 pose, maka matrik data pelatihan yang terbentuk adalah 200x165.000. ini berarti akan didapatkan matrik *eigenfaces* sebesar 200x200 (200 data pelatihan, masing-masing data mempunyai ciri sebanyak 200).
b. Nilai rata-rata yang dimaksudkan adalah rata-rata keseluruhan citra. Jika dimensi matrik data pelatihan yang dibentuk adalah 200x165.000, maka dimensi dari rata-rata citra adalah 1x165.000, dengan demikian bisa dilihat bentuk rata-rata citra data pelatihan yang digunakan dengan mengembalikan dimensi 1x165.000 menjadi 150x110.
c. Melakukan proyeksi data setelah didapatkan nilai *eigenvector* yang telah diurutkan nilai *eigenvalue*-nya.

Algoritma ekstraksi fitur menggunakan PCA : [PUR-10:223]

1. Masing-masing bentuk gambar 2-dimensi dijadikan 1 dimensi (disusun menjadi *matrix* 1 baris)
2. Cari nilai rata-rata seluruh citra menggunakan persamaan 2.1:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_5 \end{bmatrix} = \begin{bmatrix} x_{1,1}x_{1,2}x_{1,3} & \dots & x_{1,n} \\ x_{2,1}x_{2,2}x_{2,3} & \dots & x_{2,n} \\ x_{3,1}x_{3,2}x_{3,3} & \dots & x_{3,n} \\ \dots & \dots & \dots \\ x_{5,1}x_{5,2}x_{5,3} & \dots & x_{m,n} \end{bmatrix} \quad \dots (2.1)$$

Nilai rata-rata dari parameter ke-i, dari persamaan 2.2 dapat dituliskan

$$\mu_i = \frac{(x_{1,i} + x_{2,i} + x_{3,i} + \ldots + x_{m,i})}{m}$$

$$= \frac{\sum_{j=1}^{m} x_{j,i}}{m} = [\mu_1, \mu_2, \ldots, \mu_n] \quad \ldots (2.2)$$

3. Hitung nilai *zeromean* menggunakan persamaan 2.3 :

$$\emptyset = x_{j,i} - \mu_i \quad \ldots . (2.3)$$

$$\mu = \begin{bmatrix} \mu_{1,1}\mu_{1,2}\mu_{1,3} & \ldots & \mu_{1,n} \\ \mu_{2,1}\mu_{2,2}\mu_{2,3} & \ldots & \mu_{2,n} \\ \ldots \quad \ldots & \ldots & \ldots \quad \ldots \\ \mu_{m,1}\mu_{m,2}\mu_{m,3} & \ldots & \mu_{m,n} \end{bmatrix} \quad \ldots . (2.4)$$

Dan nilai dari $[\mu_{i,1}, \mu_{i,2}, \mu_{i,3}, \ldots, \mu_{i,n}]$ pada baris ke i = nilai baris ke i+1 dan berlaku $\forall j, j \in 1,2,3, \ldots, m-1$, maka hasil perhitungan matriks di atas dapat digunakan untuk menghitung *zero mean*.

4. Bentuk matrik *kovarian* pada persamaan 2.5 :

$$C = \frac{1}{m-1} . \emptyset_{j,i} . \emptyset_{j,i}^{T}$$

$$C = \frac{1}{m-1} . (x_{j,i} - \mu_i) (x_{j,i} - \mu_i)^{\mathrm{T}} \quad \ldots (2.5)$$

Keterangan :

C = nilai matrik *kovarian*

m = jumlah *sampel* keseluruhan dengan m = jumlah data pelatihan x model tiap data yang digunakan

$\emptyset_{j,i}$ = nilai *zero mean* hasil perhitungan dari persamaan 2.3

$\emptyset_{j,i}^{T}$ = nilai *zero mean* hasil perhitungan dari persamaan 2.3 yang ditranspose

5. Tentukan matrik *eigenvector* dengan persamaan :

Jika C adalah matrik bujur sangkar dengan ukuran sembarang m > 1, maka vektor tak nol Λ pada R^n di sebut *eigenvector* dari C jika Cλ suatu penggandaan skalar dari Λ, yang dihitung mengunakan persamaan di bawah ini:

$$\mathbf{C\,\Lambda = \lambda\,\Lambda} \quad \ldots (2.6)$$

Skalar λ disebut sebagai *eigenvalue* dari C dan Λ disebut sebagai *eigenvector* dari C yang berpadanan terhadap λ, untuk mendapatkan *eigenvektor* dan *eigenvalue* maka dari persamaan diatas dapat ditulis menjadi persamaan 2.7 :

$$C \Lambda = \lambda I \Lambda$$
$$(\lambda I - C) \Lambda = 0 \quad \text{... (2.7)}$$
$$\text{Det}(\lambda I - C) = 0$$

Atau dapat dituliskan mengggunakan persamaan di bawah ini :

$$\begin{bmatrix} \lambda & 0 & \dots & 0 \\ 0 & \lambda & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda \end{bmatrix} \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,8} \\ c_{2,2} & c_{2,2} & \dots & c_{2,n} \\ \dots & \dots & \dots & \dots \\ c_{m,1} & c_{m,2} & \dots & c_{m,n} \end{bmatrix} \quad \text{... (2.8)}$$

$$\begin{bmatrix} \lambda - c_{1,1} & - c_{1,2} & \dots & - c_{1,n} \\ - c_{2,2} & \lambda - c_{2,2} & \dots & - c_{2,n} \\ \dots & \dots & \dots & \dots \\ - c_{m,1} & - c_{m,2} & \dots & \lambda - c_{m,n} \end{bmatrix}$$

6. Urutkan *eigenvalue* secara *decreasing* dan dari kolom *eigenvektor* menyesuaikan hasil indek dari *eigenvalue*.
7. Bentuk data set baru/ final data/ proyeksi dengan

$$\text{NewDataSet} = (\text{Zero Mean})^T * \text{eigenvector} \quad \text{... (2.9)}$$

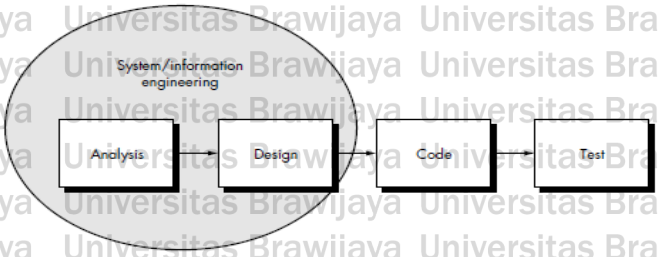
8. Hitung matrik bobot :

$$\text{MatWeight} = \text{MatrixTraining} * (\text{NewDataSet})^T \quad \text{... (2.10)}$$

2.3 Rekayasa Perangkat Lunak

Software engineering adalah penetapan dan penggunaan prinsip-prinsip *sound engineering* untuk memperoleh *software* ekonomis yang handal dan bekerja efisien pada mesin nyata. Rekayasa Perangkat Lunak: (1) Penerapan pendekatan yang sistematis dan disiplin, dihitung dengan operasi, pengembangan, dan pemeliharaan perangkat lunak, yaitu penerapan rekayasa perangkat lunak. (2) Studi tentang pendekatan seperti pada (1). Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat dari proyek dan aplikasi, metode dan alat yang digunakan, kontrol dan kiriman yang diperlukan.[PRE-01:20]

Salah satu dari model proses yang digunakan adalah *linear sequential model*. *Linear sequential model* biasa disebut sebagai *classic life cycle* atau *waterfall model*. Model proses *waterfall* ini merekomendasikan pendekatan yang sistematis dan terurut (*systematic and sequential approach*) untuk pengembangan perangkat lunak yang dimulai dari analisis kebutuhan (*requirement analysis*), perancangan (*design*), implementasi (*coding*), pengujian (*testing*), dan pemeliharaan (*maintenance*) [PRE-01:28]. Proses pengembangan menggunakan model proses *waterfall* ini terlihat pada gambar 2.5.



Gambar 2.5 *linear sequential* model

Sumber: [PRE-01:28]

Pada skripsi ini digunakan metode analisis kebutuhan, perancangan, implementasi, dan pengujian berorientasi objek menggunakan bahasa pemodelan UML (*Unified Modelling Language*). UML adalah bahasa untuk menggambarkan (*visualizing*), menspesifikasikan (specifying), membangun (*constructing*), dan mendokumentasikan (*documenting*) artefak dari sebuah sistem perangkat lunak [BOO-05].

2.3.1 Analisis Kebutuhan

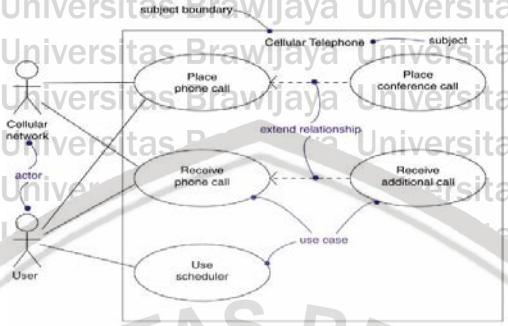
Objektif dari analisis berorientasi objek (*Object Oriented Analysis/OOA*) adalah untuk mengembangkan sebuah model yang menjelaskan perangkat lunak bekerja sesuai kebutuhan yang didefinisikan oleh *customer* [PRE-01:572]. Proses OOA tidak dimulai dengan perhatiannya terhadap objek-objek. Namun proses OOA dimulai dengan pemahaman oleh siapa sistem tersebut digunakan. Aktor sistem tersebut dapat berupa orang jika sistem tersebut merupakan *human interactive system* atau berupa mesin jika sistem tersebut dilibatkan dalam *process control* atau bahkan berupa program lain jika sistem tersebut ditujukan untuk mengkoordinasi dan mengontrol aplikasi-aplikasi lain [PRE-01:581]. Diagram pemodelan aplikasi keseluruhan berdasarkan analisis kebutuhan yang dilakukan digambarkan dengan *use case diagram*.

Use case diagram memodelkan sistem dari perspektif *end-user*. Selama penggalian kebutuhan sistem, *use case* harus mampu mencapai beberapa objektif. Objektif-objektif tersebut antara lain [PRE-01:581]:

- *Use case* mampu untuk mendefinisikan kebutuhan yang bersifat fungsional dan operasional sistem dengan cara mendefinisikan skenario yang disetujui oleh *end user* dan *software engineer team*.

- *Use case* harus menyediakan penjelasan yang jelas dan tidak ambigu tentang cara *end-user* berinteraksi dengan sistem.
- *Use case* harus menyediakan dasar untuk *validation testing*.

Contoh *use case diagram* diperlihatkan pada Gambar 2.6.



Gambar 2.6 Contoh *use case diagram*

Sumber: [BOO-05]

Use case diagram terdiri atas *subject, use cases, actors, dependency, generalization*, dan *association relationship*. *Subject* adalah *class* yang dideskribsikan oleh *use cases*. *Use cases* merepresentasikan *behavior* dari *class*. *Actor* menggambarkan peran yang dilakukan oleh menusia, *hardware device*, atau sistem lain yang berinteraksi dengan sistem tersebut. Setiap *use case* harus memiliki nama (*name*) yang digunakan untuk membedakan dengan *use case* lainnya [BOO-05].

2.3.2 Perancangan

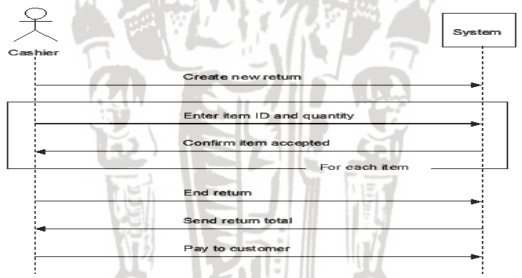
Perancangan berorientasi objek (*Objek Oriented Design/OOD*) mentransformasikan model yang dibuat menggunakan OOA (*Object Oriented Analysis*). OOD membutuhkan definisi sebuah *multilayered software architecture*, spesifikasi subsistem yang menunjukkan fungsi-fungsi yang dibutuhkan dan menyediakan dukungan infrastruktur, deskripsi dari objek-objek yang membangun sistem, serta deskripsi dari mekanisme komunikasi yang membolehkan aliran data antar *layer*, subsistem, dan objek-objek. OOD dibagi dalam dua aktivitas besar yaitu, *system design* dan *object design*. *Systemdesign* membuat *product architecture*, mendefinisikan *layer-layer* yang melakukan fungsi sistem tertentu dan mengidentifikasi kelas-kelas yang dienkapsulasi oleh subsistem yang berada pada setiap *layer*. Sebagai tambahan *system design* berhubungan dengan tiga spesifikasi komponen yaitu,

user interface,data management functions, dan *task management facilities*. Sedangkan *object design* bertumpu pada detail *internal* individu-individu kelas, mendefinisikan atribut-atribut, operasi-operasi dan *message* [PRE-01:603].

Pada tahap perancangan di skripsi ini, digunakan pemodelan dengan menggunakan *sequence diagram* dan *class diagram*.

- **Diagram Sekuensial (*Sequence Diagram*)**

Secara umum diagram sekuensial mengilustrasikan interaksi antar objek. Ketika digunakan untuk menentukan *behaviour* pada *high-level system*, diagram sekuensial menggambarkan komunikasi antara *actor* dan sistem dibawah skenario sebagai suatu urutan dari kejadian atau komunikasi. Gambar 2.8 merupakan contoh dari diagram sekuensial. Pada bagian atas merupakan objek yang terlibat yaitu *actor* dan sistem. Garis putus menggambarkan komunikasi antara keduanya. Garis panah menggambarkan pesan yang diminta oleh *actor* dan pesan balasan dari sistem. Alur tindakan digambarkan mulai dari atas ke bawah [KUR-08:252].



Gambar 2.8 Contoh diagram sekuensial

Sumber: [KUR-08:253]

2.3.3 Implementasi

Implementasi perangkat lunak dilakukan untuk merealisasikan desain dari perangkat lunak menggunakan bahasa pemrograman berorientasi objek (*Object Oriented Programming Languages/OOP*). Bahasa pemrograman berorientasi objek yang digunakan pada skripsi ini adalah *Java*.

2.3.4 Pengujian (*Testing*)

Arsitektur dari perangkat lunak berorientasi objek menghasilkan sekumpulan *layered subsystems* yang mengenkapsulasi kelas-kelas yang berkolaborasi. Setiap elemen sistem (subsistem dan *class*) melakukan fungsi yang membantu untuk mencapai kebutuhan sistem. Hal ini sangat penting untuk menguji sebuah OO *system* pada berbagai macam level yang berbeda dalam sebuah usaha untuk menemukan kesalahan-kesalahan yang mungkin terjadi dari kolaborasi kelas-kelas dan komunikasi subsistem melewati *architetural layer* [PRE-01:631].

- **Teknik Pengujian**

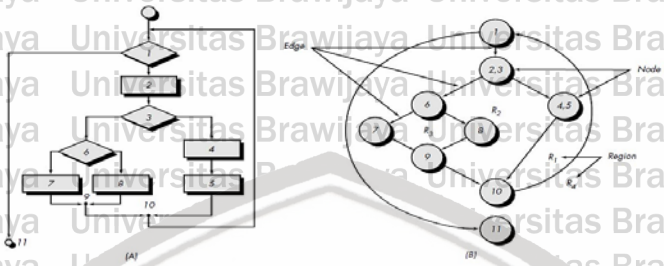
Pengujian perangkat lunak memerlukan perancangan kasus uji (*test case*) agar dapat menemukan kesalahan dalam waktu singkat dan usaha minimum. Berbagai macam metode perancangan kasus uji telah berevolusi. Metode-metode ini menyediakan *developer* pendekatan sistematis untuk pengujian. Terlebih lagi metode-metode ini menyediakan mekanisme yang dapat membantu memastikan kelengkapan dari pengujian dan menyediakan kemungkinan tertinggi untuk menemukan kesalahan-kesalahan dalam perangkat lunak [PRE-01:443]. Teknik atau metode perancangan kasus uji yang digunakan adalah *white-box testing* dan *black-box testing*.

❖ ***White-Box Testing***

White-box testing atau *glass-box testing* merupakan sebuah metode perancangan kasus uji yang menggunakan struktur kontrol dari perancangan prosedural untuk memperoleh kasus uji [PRE-01:444]. Ada dua jenis pengujian yang termasuk *white-box testing* yaitu *basis path testing* dan *control structure testing*.

Pada skripsi ini dilakukan pengujian menggunakan *basis path testing* yang diusulkan pertama kali oleh Tom McCabe [PRE-01:445]. *Basis path testing* ini memungkinkan perancang kasus uji memperoleh ukuran kompleksitas logis dari sebuah perancangan prosedural dan menggunakan pengukuran ini sebagai pedoman untuk mendefinisikan *basis set* dari jalur eksekusi (*execution path*). *Test case* yang dilakukan untuk menggunakan *basis set* tersebut dijamin untuk menggunakan setiap *statement* di dalam program paling tidak sekali selama pengujian. Sebelum metode *basis path* dapat diperkenalkan, notasi sederhana

untuk representasi aliran kontrol yang disebut diagram alir (*flow graph*) harus diperkenalkan. Setiap representasi desain prosedural yang berupa *flow chart* dapat diterjemahkan ke dalam *flow graph*. Gambar 2.9 menunjukkan transformasi *flow chart* ke *flow graph*. Setelah *flow graph* didefinisikan maka harus ditentukan ukuran kompleksitas (*cyclomatic complexity*).



Gambar 2.9 Transformasi *flow chart* ke *flow graph*

Sumber: [PRE-01:447]

Cyclomatic complexity adalah metriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila metriks ini digunakan dalam konteks metode pengujian *basis path*, maka nilai yang terhitung untuk *cyclomatic complexity* menentukan jumlah jalur independen (*independent path*) dalam basis set suatu program dan memberi batas atas bagi jumlah pengujian yang diharus dilakukan untuk memastikan bahwa semua *statement* telah dieksekusi sedikitnya satu kali.

Jalur independen adalah jalur yang melalui program yang mengenalkan sedikitnya satu rangkaian statement proses baru atau suatu kondisi baru. Untuk menentukan *cyclomatic complexity* bisa dilakukan dengan beberapa cara, diantaranya [PRE-01:448]:

i. Jumlah region pada *flow graph* sesuai dengan *cyclomatic complexity*.
ii. *Cyclomatic complexity* V(G), untuk grafik G adalah V(G) = E – N + 2, dimana E adalah jumlah *edge*, dan N adalah jumlah *node*.
iii. V(G) = P + 1, dimana P adalah jumlah *predicate node* yaitu *node* yang merupakan kondisi (ada 2 atau lebih *edge* akan keluar *node* ini).

❖ ***Black-Box Testing***

Black-box testing atau *behavioral testing* berfokus pada persyaratan fungsional perangkat lunak [PRE-01:459]. Dengan demikian, pengujian *black-box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi *input* yang sepenuhnya

menggunakan semua persyaratan fungsional untuk semua program. Pengujian *black-box* bukan merupakan alternatif dari teknik *white-box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white-box*.

Pengujian *black-box* berusaha menemukan kesalahan dalam kategori berikut [PRE-01:460]:

- fungsi-fungsi yang tidak benar atau hilang
- kesalahan *interface*
- kesalahan dalam struktur data atau akses *database* eksternal
- kesalahan kinerja
- inisialisasi dan kesalahan terminasi

Tidak seperti pengujian *white-box*, yang dilakukan pada saat awal proses pengujian, pengujian *black-box* cenderung diaplikasikan selama tahap akhir pengujian. Karena pengujian *black-box* memperhatikan struktur kontrol, maka perhatian berfokus pada domain informasi. Pengujian didesain untuk menjawab pertanyaan-pertanyaan berikut [PRE-01:460]:

- Bagaimana validitas fungsional diuji ?
- Kelas *input* apa yang akan membuat *test case* menjadi baik?
- Apakah sistem sangat sensitif terhadap harga *input* tertentu?
- Bagaimana batasan dari suatu data diisolasi ?
- Kecepatan dan volume data apa yang dapat ditolerir oleh sistem ?
- Apa pengaruh kombinasi tertentu dari data terhadap operasi sistem ?

- **Strategi Pengujian**

Strategi untuk pengujian perangkat lunak mengintegrasikan metode desain *test case* perangkat lunak ke dalam sederetan langkah yang direncanakan dengan baik, dan hasilnya adalah konstruksi perangkat lunak yang berhasil [PRE-01:477]. Sejumlah strategi pengujian perangkat lunak telah diusulkan di dalam literatur. Strategi pengujian harus mengakomodasi pengujian tingkat rendah yang diperlukan untuk membuktikan bahwa segmen kode sumber yang kecil telah diimplementasikan dengan tepat, demikian juga pengujian tingkat tinggi

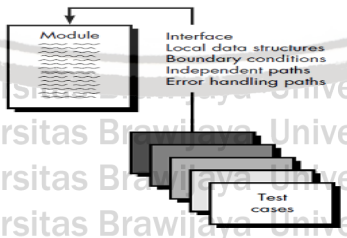
yang memvalidasi fungsi-fungsi sistem *mayor* yang berlawanan dengan kebutuhan pelanggan.

Proses pengujian dimulai dengan pengujian yang berfokus pada setiap modul secara individual (*unit testing*), dilanjutkan dengan pengujian integrasi (*integration testing*) dan berakhir pada pengujian validasi (*validation testing*) [PRE-01:481].

❖ **Unit Testing**

Pengujian unit berfokus pada usaha verifikasi pada inti terkecil dari desain perangkat lunak, yakni modul. Dengan menggunakan gambaran desain prosedural sebagai panduan, jalur kontrol yang penting diuji untuk mengungkap kesalahan di dalam batas modul tersebut. Kompleksitas relatif dari pengujian dan kesalahan yang diungkap dibatasi oleh ruang lingkup batasan yang dibangun untuk pengujian unit. Pengujian unit biasanya berorientasi pada *white-box,* dan langkahnya dapat dilakukan secara paralel untuk model bertingkat [PRE-01:485].

Pengujian yang terjadi sebagai bagian dari unit digambarkan secara skematis pada Gambar 2.10. Interface modul diuji untuk memastikan bahwa informasi secara tepat mengalir masuk dan keluar dari inti program yang diuji. Struktur data lokal diuji untuk memastikan bahwa data yang tersimpan secara temporal dapat tetap menjaga integritasnya selama semua langkah di dalam suatu algoritma dieksekusi. Kondisi batas diuji untuk memastikan bahwa modul beroperasi dengan tepat pada batas yang ditentukan untuk membatasi pemrosesan. Semua jalur independen (jalur dasar) yang melalui struktur kontrol dipakai sedikitnya satu kali. Dan akhirnya, penanganan kesalahan uji [PRE-01:485].



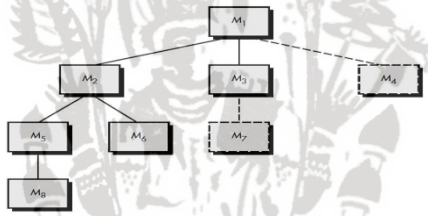
Gambar 2.10 *Unit Testing*

Sumber: [PRE-01:487]

❖ **Integration Testing**

Pengujian integrasi adalah teknik sistematis untuk mengkonstruksi struktur program sambil melakukan pengujian untuk mengungkap kesalahan sehubungan dengan interfacing. Sasarannya adalah untuk mengambil modul yang dikenai pengujian unit dan membangun struktur program yang telah ditentukan oleh desain. *Integration testing* berorientasi *black box* dan mempunyai dua pola pengujian yaitu integrasi *top-down* (*top-down integration*) dan integrasi *bottom-up* (*bottom-up integration*) [PRE-01:488].

Integrasi *top-down* adalah pendekatan inkremental terhadap struktur program. Modul diintegrasikan dengan menggerakkan ke bawah melalui hirarki kontrol, dimulai dengan modul kontrol utama (program utama). Subordinat program terhadap modul kontrol utama digabungkan ke dalam struktur dengan cara *depth-first* atau *breadth-first* [PRE-01:489]. Gambar 2.11 menunjukkan pola pengujian integrasi *top-down*.



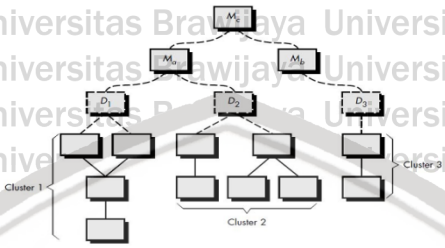
Gambar 2.11 Integrasi *top-down*

Sumber:[PRE-01:489]

Proses integrasi *top-down* dilakukan dalam lima langkah [PRE-01:489]:

- Modul kontrol utama digunakan sebagai *test driver* dan *stub* digunakan untuk menggantikan semua komponen dibawahnya.
- Pemilihan pendekatan integrasi yang diinginkan (*depth atau breadth first*).
- Pengujian dikerjakan untuk setiap komponen yang diintegrasikan.
- *Stub* digantikan dengan komponen yang sebenarnya setelah menyelesaikan serangkaian pengujian.
- Proses akan terus dilakukan sampai membentuk sebuah perangkat lunak yang utuh.

Pengujian integrasi *bottom-up* memulai konstruksi dan pengujian dengan modul atomik (modul pada tingkat paling rendah pada struktur program). Hal ini terlihat pada Gambar 2.12. Karena modul diintegrasikan dari bawah keatas, maka pemrosesan yang diperlukan untuk modul subordinat ke suatu tingkat yang diberikan akan selalu tersedia dan kebutuhan akan *stub* dapat dieliminasi [PRE-01:490].



Gambar 2.12 Integrasi *bottom-up*

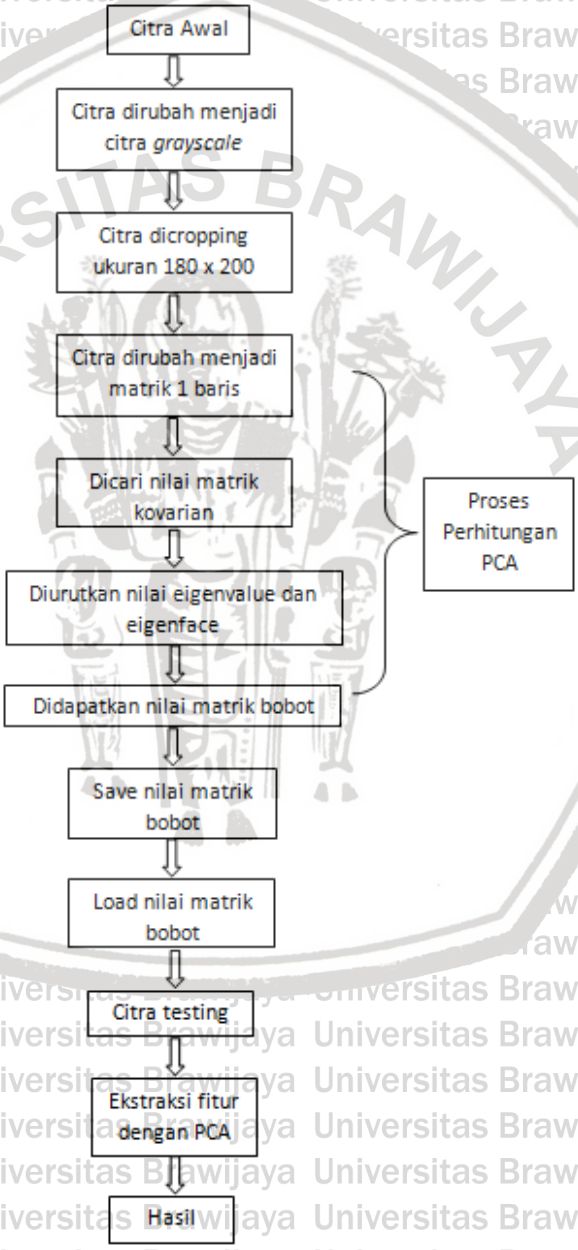
Sumber: [PRE-01:491]

Integrasi *bottom-up* dapat diimplementasi dengan langkah-langkah berikut [PRE-01:490]:

- Komponen pada level terendah digabung ke dalam sebuah sub fungsi (*cluster*).
- *Driver* akan dibuat untuk menguji setiap *cluster*.
- *Driver* akan diganti dengan modul sesungguhnya setelah sub fungsi teruji.
- Proses akan terus dilakukan sampai membentuk sebuah perangkat lunak yang utuh.

BAB III
METODOLOGI PENELITIAN

Pada bab ini dijelaskan langkah-langkah yang akan dilakukan dalam perancangan, implementasi dan pengujian dari aplikasi perangkat lunak yang akan dibuat. Kesimpulan dan saran disertakan sebagai catatan atas aplikasi dan kemungkinan arah pengembangan aplikasi selanjutnya.



Gambar 3.1 Alur Sistem Prediksi Citra

Sumber : Metodologi

Penjelasan alur sistem aplikasi prediksi citra untuk mendeteksi hama dan penyakit tanaman jagung sebagai berikut :

1. Citra awal

   Foto hama jagung yang belum dilakukan proses pengolahan citra.
2. Citra dirubah menjadi citra *grayscale*

   Citra hama yang akan diprediksi harus dirubah menjadi citra *grayscale* untuk memudahkan dalam proses pengolahan citra
3. Citra dicropping ukuran 180 x 200

   Citra hama yang telah dirubah menjadi grayscale dilakukan proses pemotongan sesuai dengan bentuk hama menjadi ukuran 180 x 200
4. Citra dirubah menjadi *matrix* 1 baris

   Nilai *matrix* citra hama dirubah menjadi nilai *matrix* 1 baris untuk memudahkan mencari nilai rata-rata *matrix* data training.
5. Dicari nilai matriks kovarian

   Nilai matriks kovarian didapatkan dengan menghitung nilai transpose matriks dari *zero mean*. Hasil nilai matriks kovarian digunakan sebagai acuan jumlah matriks yang digunakan sebagai matriks bobot proses PCA.
6. Diurutkan nilai *eigenvalue* dan *eigenface*

   Setelah proses matrik kovarian kemudian dicari nilai *eigenvalue* dan *eigenface* kemudian diurutkan nilai *eigenvalue* secara *decreasing* dan kolom dari *eigenvector* menyesuaikan hasil dari indek dari *eigenvalue*.
7. Didapatkan nilai matrik bobot

   Merupakan proses perhitungan akhir dari proses PCA karena matriks bobot yang digunakan proses pencocokan dengan matrik bobot data tes.
8. *Load* nilai matriks bobot

   Proses untuk memuat nilai matriks bobot untuk proses tes PCA
9. Citra testing

   Proses awal untuk memasukkan citra tes
10. Ekstraksi fitur dengan PCA

    Merupakan proses ekstraksi fitur dengan PCA untuk citra tes yang diuji. Nilai matriks bobot data tes yang didapatkan kemudian dicari nilai matriks bobot yang hampir sama dengan data uji lalu ditampilkan citra uji yang sesuai.
11. Hasil

    Menampilkan hasil akhir prediksi citra PCA.

3.1 Studi Literatur

Studi literatur menjelaskan kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

a. Penggunaan citra digital dalam format JPEG sebagai *input* untuk proses pengolahan citra digital yang diolah sehingga mendapatkan *output* yang sesuai dengan data *training*.
b. Pengolahan Citra Digital
c. Ekstraksi Fitur : *PrincipalComponentAnalysis* (PCA)
d. Rekayasa Perangkat Lunak
   - Analisis Kebutuhan (*RequirementAnalysis*)
   - Perancanga (*Design*)
   - Implementasi
   - Pengujian (*Testing*)
     - Teknik Pengujian :
       - *White-Box Testing*
       - *Black-Box Testing*
     - Strategi Pengujian :
       - *Unit Testing*
       - *Integration Testing*
       - *Validation Testing*

3.2 Pengembangan Perangkat Lunak

Tahap pengembangan perangkat lunak dalam pembuatan aplikasi ini menggunakan *classic life style* atau yang lebih dikenal dengan istilah *waterfall*.

3.2.1 Analisis Kebutuhan

Analisis kebutuhan dilakukan dengan menentukan kebutuhan apa saja yang dibutuhkan untuk membangun sistem pengolahan citra digital. Metode analisis yang digunakan adalah *Object Oriented Analysis* dengan menggunakan bahas pemodelan UML (*Unified Modelling Language*). Diagram *use case* digunakan untuk mengetahui jalannya sistem sebagai respon dari *request* yang berasal dari luar sistem dan bagaimana *actor* berinteraksi dengan sistem. Skenario yang akan dijalankan dalam skenario ini adalah:

- *User* melakukan *updatedatabase* kemudian memasukkan *input* berupa gambar hama jagung
- Sistem akan memproses masukkan dari *user* dengan menggunakan pengolahan citra *digital* menggunakan ekstraksi fitur *PrincipalComponentAnalysis* (PCA) kemudian hasilnya ditampilkan jika sesuai dengan data *training*.
- *User* akan menerima *output* berupa gambar yang sesuai dengan data *test*.

Kebutuhan yang digunakan dalam pembuatan sistem pengolahan citra digital ini meliputi:

1. Kebutuhan *Hardware*, meliputi:
   - Komputer PC atau *notebook*
2. Kebutuhan *Software*, meliputi:
   - *MicrosoftWindowsXP* maupun *Windows7* sebagai sistem operasi
   - *Netbeans* 6.9 sebagai *Integrated Development Environtment*
   - JSE (*Java Standard Edition*) 6 sebagai platform pengembangan
   - *VisualParadigm* sebagai tool pembuatan diagram pemodelan sistem
3. Data yang dibutuhkan meliputi:
   - *Data training* yang diperoleh dari pengolahan citra digital hama dan penyakit jagung
   - Data masukan yaitu data citra digital hama dan penyakit jagung

3.3 Perancangan

Setelah menentukan kebutuhan untuk membangun sistem, maka langkah selanjutnya yaitu desain atau perancangan sistem. Perancangan aplikasi berdasarkan *Object Oriented Analysis* dan *Object Oriented Design* yaitu menggunakan pemodelan UML (*Unified Modeling Language*). Pada tahap perancangan di penelitian ini, digunakan pemodelan dengan menggunakan dua macam diagram, yaitu *class diagram* dan *sequence diagram*. Perancangan kelas-kelas dan *interface-interface* yang dibutuhkan dimodelkan dalam *class diagram*. Hubungan interaksi antar elemen (objek) yang telah diidentifikasi, dimodelkan dalam *sequence diagram* yang menggambarkan interaksi antar objek yang disusun dalam urutan waktu.

3.4 Implementasi

Mengacu pada desain sistem, maka implementasi sistem dilakukan dengan pembangunan sistem. Implementasi perangkat lunak dilakukan dengan menggunakan bahasa pemrograman berorientasi objek yaitu menggunakan bahasa pemrograman Java (*Java Standard Edition*). Implementasi sistem pengolahan citra digital untuk mendeteksi hama dan penyakit jagung meliputi:

- Pembuatan *user interface* dan penerapan algoritma ekstraksi fitur *PrincipalComponentAnalysis*.
- Memasukkan data citra digital kemudian dilakukan proses ekstraksi fitur PCA.

3.5 Pengujian

Pengujian perangkat lunak pada penelitian ini dilakukan agar dapat menunjukkan bahwa perangkat lunak telah mampu bekerja sesuai dengan spesifikasi dari kebutuhan yang melandasinya. Pengujian yang dilakukan meliputi pengujian performa sistem yang meliputi pengujian perangkat lunak serta pengujian validitas sistem. Strategi pengujian perangkat lunak yang digunakan yaitu pengujian unit (*unit testing*), dan pengujian validasi (*validation testing*). Sedangkan teknik atau metode pengujian yang digunakan adalah metode pengujian *white box* dan *black box*.

Pada penelitian ini akan dilakukan pengujian perangkat lunak sistem pengolahan citra digital untuk mendeteksi hama dan penyakit jagung. Pengujian dimulai dari pengujian unit, kemudian dilanjutkan dengan pengujian integrasi, dan berakhir dengan pengujian validasi. Pada tahap pengujian unit dan pengujian integrasi digunakan metode pengujian *whitebox* dengan teknik *basispath*. Pada tahap pengujian validasi digunakan teknik *blackbox*. Pengujian yang dilakukan yaitu pengujian kualitas pada citra *digital*.

3.6 Pengambilan Kesimpulan dan Saran

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian sistem aplikasi telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

BAB IV
PERANCANGAN

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan yang dilakukan meliputi dua tahap. Proses analisis kebutuhan dilakukan pada tahap pertama dan tahap kedua adalah proses perancangan perangkat lunak. Tahap analisis kebutuhan terdiri dari dua langkah yaitu membuat daftar kebutuhan *user* dan menggunakan pemodelan *use case diagram* untuk menggambarkan kebutuhan tersebut. Proses perancangan perangkat lunak mempunyai dua tahap, yaitu perancangan umum dan perancangan detail aplikasi pengolahan citra digital untuk mendeteksi hama dan penyakit pada tanaman jagung. Perancangan umum menggambarkan relasi antar paket (*package*) dan kelas (*class*) sebagai pemodelan sistem secara keseluruhan. Perancangan detail menggunakan *class diagram* dan *sequence diagram* sebagai pemodelan perangkat lunak.

4.1 **Analisis Kebutuhan**

Pada analisis kebutuhan ini diawali dengan identifikasi aktor yang terlibat dengan sistem, penjabaran kebutuham dan kemudian memodelkannya ke dalam suatu *use case diagram.* Analisis kebutuhan ini ditujukan untuk menggambarkan kebutuhan-kebutuhan yang harus disediakan oleh sistem agar dapat memenuhi kebutuhan pengguna.

4.1.1 Gambaran Umum Perangkat Lunak

Pembahasan gambaran umum perangkat lunak prediksi citra terdiri dari atas dua bagian, yaitu deskripsi umum perangkat lunak citra dan cara penggunaan perangkat lunak citra.

4.1.1.1 Deskripsi Perangkat Lunak Prediksi Citra

Perangkat lunak yang dikembangkan dalam proyek skripsi ini adalah perangkat lunak Prediksi Citra. Perangkat lunak Prediksi Citra adalah perangkat lunak yang dapat digunakan untuk memprediksi hama pada tanaman jagung. Perangkat lunak Predeksi Citra diharapkan mampu membantu dalam melakukan predeksi hama tanaman jagung.

Perangkat lunak Prediksi Citra memiliki dua bagian yaitu subsistem *administrator* dan subsistem *user*.

a. Aplikasi *user*

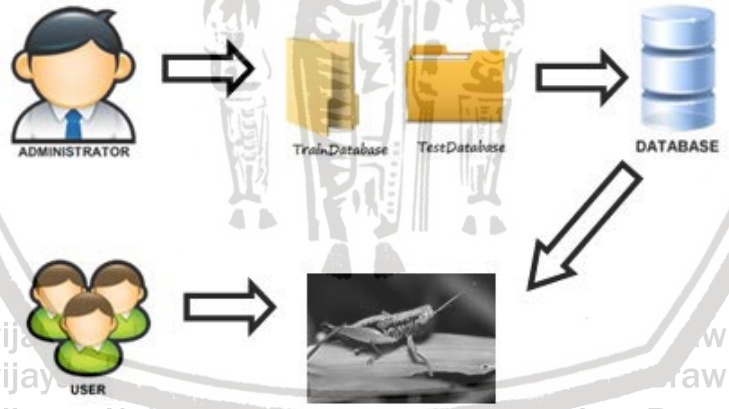
Aplikasi *user* dari perangkat lunak Prediksi Citra adalah aplikasi citra digital yang dapat mengolah citra *test* hama tanaman jagung dan menampilkan output berupa citra yang sama dengan data *training*.

b. Aplikasi *administrator*

Aplikasi *administrator* dari perangkat lunak Prediksi Citra adalah aplikasi yang digunakan untuk mengolah elemen perangkat lunak lebih lanjut. Proses yang terdapat dalam aplikasi administrator adalah memasukkan citra digital hama pada folder *TestDatabase* dan *TrainDatabase*.

4.1.1.2 Cara Penggunaan Perangkat Lunak Citra

Perangkat lunak prediksi citra memiliki urutan langkah kerja seperti berikut yang ditunjukkan pada gambar 4.1 di bawah ini :



Gambar 4.1 Urutan Langkah kerja perangkat lunak Prediksi Citra

Sumber: Perancangan

4.1.2 Identifikasi Aktor

Tahap ini mempunyai tujuan untuk melakukan identifikasi terhadap aktor yang akan berinteraksi dengan sistem. Tabel 4.1 memperlihatkan sebuah aktor beserta penjelasannya yang merupakan hasil dari proses identifikasi aktor.

Tabel 4.1 Identifikasi Aktor

Aktor	Deskripsi Aktor
Administrator	Admin merupakan aktor pengguna yang bertugas untuk memasukkan data citra *test* dan *training*.
User	User merupakan aktor pengguna yang menggunakan sistem untuk mendeteksi hama dan penyakit pada tanaman jagung.

Sumber: Perancangan

4.1.3 Analisis Data

Analisis data bertujuan untuk mendapatkan struktur penyimpanan data yang dibutuhkan perangkat lunak prediksi citra. Struktur penyimpanan data perangkat lunak prediksi citra disusun berdasarkan analisis data sebagai berikut :

a. Data *test* yang terdiri dari data citra hama yang jelas bentuknya atau 2-dimensi, misalnya belalang, kumbang bubuk, lalat bibit, tikus, kutu daun, penggerek tongkol jagung, penggerek batang jagung, ulat grayak, ulat jengkal jagung.
b. Data *training* yang terdiri atas data citra hama yang digunakan adalah citra yang posisi hama hampir sama tetapi memiliki bentuk yang sedikit berbeda.
c. Jumlah data tes yang digunakan adalah 9 citra dan jumlah data pengujian yang digunakan 15 citra.
d. Jumlah data training yang digunakan adalah 10 citra.
e. Nilai Pixel yang digunakan sebagai matriks bobot PCA adalah 54 piksel. Dengan perhitungan :

- Menggunakan 6 klas data, yaitu : belalang, kutu bubuk, ulat grayak, ulat jengkal jagung, penggerek batang jengkal jagung, penggerek batang jagung, dan tikus.
- Nilai variabel m = 6 * 10 = 60.
- Nilai piksel citra 180 * 200 = 36000
- Maka nilai matrik kovarian (persamaan 2.5) yang digunakan sebagai acuan jumlah matriks bobot adalah (60 * 36000) x (36000 * 60) = 60 piksel.

4.1.4 Daftar Kebutuhan

Daftar kebutuhan ini terdiri dari sebuah kolom yang menguraikan kebutuhan yang harus disediakan oleh sistem, dan pada kolom yang lain akan menunjukkan nama *use case* yang akan menunjukkan fungsionalitas masing-masing kebutuhan tersebut. Daftar kebutuhan fungsional dan non fungsional keseluruhan sistem ditunjukkan pada Tabel 4.2.

Tabel 4.2 Daftar Kebutuhan Fungsional *Administrator*

Nomor SRS	Kebutuhan	Nama *Use Case*
SRS_001_01	Sistem harus menyediakan antarmuka untuk melakukan *update database*	Memperbarui *database*

Sumber: Perancangan

Tabel 4.3 Daftar Kebutuhan Fungsional *User*

Nomor SRS	Kebutuhan	*Use Case*
SRS_002_01	Sistem harus menyediakan antarmuka untuk melakukan tes	Melakukan Tes
SRS_002_02	Sistem harus menyediakan antarmuka untuk melakukan tes citra *gray*	Melakukan tes citra *gray*
SRS_002_03	Sistem harus menyediakan antarmuka untuk menampilkan nilai *pixel* citra	Menampilkan nilai *pixel* citra
SRS_002_04	Sistem harus menyediakan antarmuka untuk menyimpan citra	Menyimpan citra

Sumber: Perancangan

Daftar kebutuhan non-fungsional perangkat lunak prediksi citra diperlihatkan pada Tabel 4.3.

Tabel 4.4 Daftar Kebutuhan Non-Fungsional

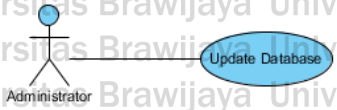
Parameter	Deskripsi Kebutuhan
Avaliability	Aplikasi ini harus dapat beroperasi selama waktu yang ditentukan.
Response Time	Aplikasi ini harus cepat dalam melakukan proses *updatedatabase* dan testing.
Memory	Aplikasi ini harus ringan dan tidak membutuhkan *memory* yang besar.

Sumber: Perancangan

4.1.5 Diagram *Use Case*

Kebutuhan-kebutuhan fungsional yang diperlukan oleh *user* dan harus disediakan oleh sistem akan dimodelkan dalam diagram *use case*. Secara keseluruhan sistem ini memiliki 4 buah *use*

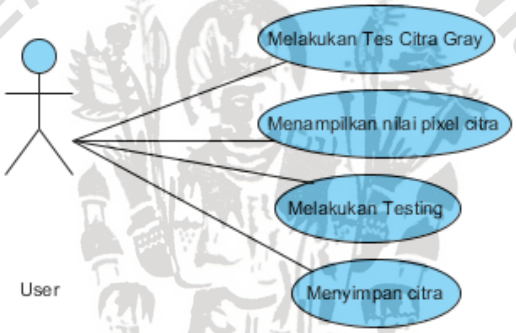
case, yaitu memperbarui data *training*, memperbarui *datatest*, memperbarui *database*, dan melakukan *test*.



Gambar 4.2 Diagram *Use Case* untuk *Administrator*

Sumber: Perancangan

Diagram *use case* untuk *user* ditunjukkan pada Gambar 4.3.



Gambar 4.3 Diagram *Use Case* untuk *User*

Sumber: Perancangan

4.1.6 Skenario *Use Case*

Secara lebih mendetail, masing-masing *use case* yang terdapat pada diagram *use case*, dijabarkan dalam skenario *use case*. Di dalam skenario *use case*, akan diberikan uraian nama *use case*, aktor yang berhubungan dengan *use case* tersebut, deskripsi global tentang *use case*, pra-kondisi yang harus dipenuhi dan kondisi akhir yang diharapkan setelah berjalannya fungsional *use case*. Sistem juga akan diberikan ulasan yang berkaitan dengan tanggapan dari sistem atas suatu aksi yang diberikan oleh aktor (aliran utama), serta kejadian alternatif.

Kebutuhan fungsional yang harus disediakan oleh sistem adalah kebutuhan memperbarui *database*. Kebutuhan tersebut direpresentasikan oleh *use case* Memperbarui *Database*. Tabel 4.5 merupakan skenario *use case* Memperbarui *Database*.

Tabel 4.5 Skenario *Use Case* Memperbarui *Database*

Identifikasi	
Nomor Use Case	SRS_001_03
Nama	Memperbarui *Database*
Tujuan	Untuk melakukan perbaruan *database*
Deskripsi	*Use case* ini menjelaskan proses untuk memperbarui *database*.
Aktor	*Administrator*
Skenario	
Kondisi Awal	Sistem menampilkan halaman awal aplikasi.
Aksi Aktor	**Reaksi Sistem**
1. ***Administrator* memilih menu "*Format*"**	2. Sistem menampilkan pilihan menu yang ada pada *Format*.
3. ***Administrator*memilih sub-menu *UpdateDatabase***	4. Sistem menampilkan halaman "Pilih Direktori" yang akan diperbarui *database*.
	5. Sistem menampilkan data *image* yang telah diperbarui.
Kondisi Akhir	Data citra berhasil diperbarui *database*.

Sumber: Perancangan

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melakukan tes. Kebutuhan tersebut direpresentasikan oleh *use case* Melakukan Tes. Tabel 4.6 merupakan skenario *use case* Melakukan Tes.

Tabel 4.6 Skenario *Use Case* Melakukan Tes

Identifikasi	
Nomor Use Case	SRS_002_01
Nama	Melakukan Tes
Tujuan	Untuk melakukan testing aplikasi.
Deskripsi	*Use case* ini menjelaskan proses

	bagaimana cara proses prediksi citra.
Aktor	*User*
Skenario	
Kondisi Awal	Sistem menampilkan halaman awal aplikasi.
Aksi Aktor	**Reaksi Sistem**
1. *User* memilih menu File	2. Sistem menampilkan pilihan menu yang ada pada *File*.
3. *User* memilih sub-menu *Open* kemudian memilih gambar yang akan dilakukan tes.	4. Sistem menampilkan data *image* yang telah dipilih untuk tes data *image*.
5. *User* memilih menu "*Format*"	6. Sistem menampilkan pilihan menu yang ada pada *Format*.
7. *User* menekan tombol "*Recognition* PCA"	8. Sistem akan memproses data *image* menggunakan PCA.
	9. Sistem menampilkan *output* data tes hasil PCA
Kondisi Akhir	Data citra berhasil untuk dilakukan "*Recognition* PCA" dan hasilnya dapat ditampilkan pada sistem.

Sumber: Perancangan

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melakukan tes citra *gray*. Kebutuhan tersebut direpresentasikan oleh *use case* Melakukan Tes Citra *Gray*. Tabel 4.7 merupakan skenario *use case* Melakukan Tes Citra *Gray*.

Tabel 4.7 Skenario *Use Case* Melakukan Tes Citra *Gray*

Identifikasi	
Nomor Use Case	SRS_002_02
Nama	Melakukan Tes Citra *Gray*
Tujuan	Untuk melakukan tes citra *gray*.
Deskripsi	*Use case* ini menjelaskan proses bagaimana cara proses testing citra *gray*.

Aktor	*User*
Skenario	
Kondisi Awal	Sistem menampilkan halaman awal aplikasi.
Aksi Aktor	**Reaksi Sistem**
1. *User* memilih menu *File*	2. Sistem menampilkan pilihan menu yang ada pada *File*.
3. *User* memilih sub-menu *Open* kemudian memilih gambar yang akan dilakukan tes.	4. Sistem menampilkan data *image* yang telah dipilih untuk tes data *image*.
5. *User* memilih menu *Format*	6. Sistem menampilkan pilihan menu yang ada pada *Format*.
7. *User* memilih *sub-menu "Gray"*	8. Sistem akan menampilkan citra hasil proses *gray*.
Kondisi Akhir	Sistem berhasil menampilkan citra berwarna *gray*.

Sumber: Perancangan

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk menampilkan nilai *pixel* citra. Kebutuhan tersebut direpresentasikan oleh *use case* Menampilkan nilai *pixel* citra. Tabel 4.8 merupakan skenario *use case* Menampilkan nilai *pixel* citra.

Tabel 4.8 Skenario *Use Case* Menampilkan Nilai *Pixel* Citra

Identifikasi	
Nomor Use Case	SRS_002_03
Nama	Menampilkan Nilai *Pixel* Citra
Tujuan	Untuk menampilkan nilai *pixel* dari citra
Deskripsi	*Use case* ini menjelaskan bagaimana sistem menampilkan nilai *pixel* citra.
Aktor	*User*

Skenario	
Kondisi Awal	Sistem menampilkan halaman awal aplikasi.
Aksi Aktor	**Reaksi Sistem**
1. *User* memilih menu *File*	2. Sistem menampilkan pilihan menu yang ada pada *File*.
3. *User* memilih *sub-menuOpen* kemudian memilih gambar yang akan dilakukan tes.	4. Sistem menampilkan data *image* yang telah dipilih untuk tes data *image*.
5. *User* memilih menu *Format*	6. Sistem menampilkan pilihan menu yang ada pada *Format*.
7. *User* memilih *sub-menu "ShowPixel"*	8. Sistem akan menampilkan nilai *pixel* citra.
Kondisi Akhir	Nilai *pixel* citra berhasil ditampilkan oleh sistem.

Sumber: Perancangan

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk menyimpan citra. Kebutuhan tersebut direpresentasikan oleh *use case* Menyimpan Citra. Tabel 4.9 merupakan skenario *use case* Menyimpan Citra.

Tabel 4.9 Skenario *Use Case* Menyimpan Citra

Identifikasi	
Nomor Use Case	SRS_002_04
Nama	Menyimpan citra
Tujuan	Untuk menyimpan citra
Deskripsi	*Use case* ini menjelaskan bagaimana sistem menyimpan citra.
Aktor	*User*
Skenario	
Kondisi Awal	Sistem menampilkan halaman awal aplikasi.
Aksi Aktor	**Reaksi Sistem**

1. *User* memilih menu File	2. Sistem menampilkan pilihan menu yang ada pada *File*.
3. *User* memilih *sub-menuOpen* kemudian memilih gambar yang akan dilakukan tes.	4. Sistem menampilkan data *image* yang telah dipilih untuk tes data *image*.
5. *User* memilih menu *Format*	6. Sistem menampilkan pilihan menu yang ada pada *Format*.
7. *User* memilih sub-menu "*Recognition* PCA"	8. Sistem akan melakukan proses PCA.
	9. Sistem menampilkan *output* citra hasil proses PCA.
10. *User* memilih menu "File"	11. Sistem menampilkan pilihan menu yang ada pada "*File*".
12. *User* memilih *sub-menu* "*Save*"	13. Sistem menampilkan antarmuka untuk menyimpan citra tes.
14. *User* memilih lokasi untuk menyimpan citra.	15. Sistem telah menyimpan citra pada lokasi yang telah dipilih *user*.
Kondisi Akhir	Citra berhasil disimpan pada lokasi yang telah dipilih oleh *user*.

Sumber: Perancangan

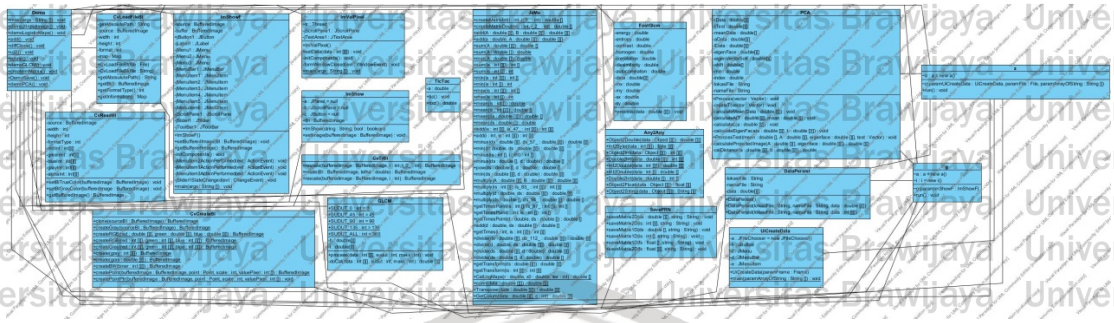
4.2 Perancangan Perangkat Lunak

Perancangan perangkat lunak dilakukan dalam empat tahap, yaitu perancangan arsitektural, pemodelan diagram *class* untuk menggambarkan perancangan struktur *class-class* yang menyusun perangkat lunak Prediksi Citra, pemodelan diagram *sequence* untuk menggambarkan interaksi antar objek atau *class* di dalam perangkat lunak Prediksi Citra. Perancanganperangkat lunak pada skripsi ini menggunakan pendekatan berorientasi objek yang direpresentasikan dengan menggunakan UML.

4.2.1 Diagram *Class*

Diagram *class* memberikan gambaran permodelan elemen-elemen *class* yang membentuk sebuah perangkat lunak. *Class* bisa didapatkan dengan menganalisis secara detail

terhadap *usecase* yang dimodelkan. Gambar 4.4 menunjukkan diagram *class* dari perangkat lunak Prediksi Citra yang akan dibuat. (Untuk gambar lebih jelas di lampiran 1)



Gambar 4.4 Diagram Kelas Sistem

Sumber: Perancangan

Keterangan penjelasan dari *class* diagram, seperti di bawah ini:

a. Klas Demo. Deskripsi atribut dan operasi dari kelas Demo diperlihatkan pada tabel 4.10 berikut:

Tabel 4.10 Kelas Demo

Nama kelas: Demo
Deskripsi: Kelas Demo merupakan klas yan g terdapat beberapa metode yang digunakan untuk mengolah *image*.
Atribut
Nama Atribut: *main* Deskripsi: merupakan atribut yang digunakan untuk mendeklarasikan variabel *main*.
Operasi
Nama Operasi : demoJInputImage () Deskripsi : merupakan operasi yang digunakan untuk menampilkan gambar sebagai *input*.
Nama Operasi : demoLogisticMaps() Deskripsi : merupakan operasi yang digunakan untuk menampilkan letak *Maps* pada klas TicToc.
Nama Operasi: add()

Deskripsi: merupakan operasi tambah untuk mendaptkan nilai setData (s).
Nama Operasi: diffClone() Deskripsi: merupakan operasi yang digunakan untuk menampilkan *bufferedImage* bi2 dan bi3.
Nama Operasi: uji2() Deskripsi: merupakan operasi yang digunakan untuk melakukan *setImageBufferedImage*,bi2, dan bi3.
Nama Operasi: rotate() Deskripsi: merupakan operasi yang digunakan untuk memberikan skala ulang *image*.
Nama Operasi: createImMatrix() Deskripsi: merupakan operasi yang digunakan untuk membuat nilai *matrix red,green,blue*.
Nama Deskripsi: DemoSave() Deskripsi: merupakan operasi yang dilakukan untuk melakukan *save image*.
Nama Deskripsi: demoPCA() Deskripsi: merupakan operasi yang digunakan untuk memanggil klas PCA dan *DataPersist*.

Sumber: Perancangan

b. Klas *CvLoadFileBI*. Deskripsi atribut dan operasi dari kelas *CvLoadFileBI* diperlihatkan pada tabel 4.11 berikut:

Tabel 4.11 Kelas *CvLoadFileBI*

Nama kelas: *CvLoadFileBI*
Deskripsi: Kelas *CvLoadFileBI* merupakan klas yan g digunakan untuk menampilkan file *BufferedImage*.
Atribut
Nama Atribut: *getAbsolutePath* Deskripsi: merupakan atribut yang digunakan untuk *file input*.
Nama Atribut : *source*

Deskripsi: merupakan atribut yang digunakan untuk proses input BufferedImage.
Nama Atribut: *width* Deskripsi: merupakan atribut yang digunakan untuk proses *input* bertipe *integer*.
Nama Atribut: *height* Deskripsi : merupakan atribut yang digunakan untuk proses *input* bertipe *integer*.
Nama Atribut: format Deskripsi : merupakan atribut yang digunakan untu k proses *input* tipe format *image*.
Nama Atribut: map Deskripsi: merupakan atribut yang digunakan untuk mendapatkan *directorifile* disimpan.
Operasi
Nama Operasi : *CvLoadFileBI*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan lebar,tinggi dan format *source*.
Nama Operasi: *getAbsolutePath*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan kembali nilai *getAbsolutePath*.
Nama Operasi: *getBI*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *BufferedImage* dari data *image*.
Nama Operasi : *getWidth*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai lebar *image.*
Nama Operasi: *getHeight*() Deskripsi: merupakan operasi yang dilakukan untuk mendapatkan nilai tinggi *image*.
Nama Operasi: *getFormatType*() Deskripsi: merupakan operasi yang dilakukan untuk mendapatkan tipe *formatimage*.

Nama Operasi: *getInformation*() Deskripsi: merupakan operasi yang digunakan untuk mendaptkan informasi *mapimage*.

Sumber: Perancangan

c. Klas *CvReadBI*. Deskripsi atribut dan operasi dari kelas *CvReadBI* diperlihatkan pada tabel 4.12 berikut:

Tabel 4.12 Kelas *CvReadBI*

Nama kelas: *CvReadBI*
Deskripsi: Kelas *CvReadBI* merupakan klas yan g digunakan untuk membaca file *BufferedImage*.
Atribut
Nama Atribut: *source* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan data *BufferedImage* dari klas *CvCreateBI*.
Nama Atribut: *width* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan lebar data *bufferedimage*.
Nama Atribut :*height* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai tinggi.
Nama Atribut:*formatType* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan tipe *bufferedimage*.
Nama Atribut :*redInt* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai lebar * tinggi *pixel* merah, bertipe *integer*.
Nama atribut: *greenInt* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai lebar * tinggi *pixel* hijau, bertipe *integer*.
Nama Atribut: *blueInt* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai

lebar * tinggi *pixel* biru, bertipe *integer*.
Nama Atribut: *grayInt* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai lebar * tinggi *pixel* abu, bertipe *integer*.
Nama Atribut: *alphaInt* Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai lebar * tinggi *pixel* alpha, bertipe *integer*.
Operasi
Nama Operasi: *setBITrueColor*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *BufferedImage* dari warna yang sesungguhnya.
Nama Operasi: *setBIGrayColor*() Deskripsi:merupakan operasi yang digunakan untuk mendapatkan nilai abu-abu pada *BufferedImage*.
Nama Operasi: *getBufferedImage*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *source*.

Sumber: Perancangan

d. Klas ImShowF.

Klas *ImShowF* merupakan klas yang berisikan perintah untuk mengatur tampilan *interface* pada sistem. Pada klas ini tampilan sistem diatur menjadi satu karena dikarenakan perintah dituliskan secara manual.

e. Klas Jama

Klas JaMa merupkan klas yang berisi tentang perhitungan untuk mendapatkan nilai *matrixInteger, matrixdouble,* kemudian dilakukan metode *transpose* lalu disesuaikan dengan kolom yang ada.

f. Klas *CvCreateBI*. Deskripsi atribut dan operasi dari kelas *CvCreateBI* diperlihatkan pada tabel 4.13 berikut:

Tabel 4.13 Kelas *CvCreateBI*

Nama kelas: *CvCreateBI*
Deskripsi: Kelas *CvCreateBI* merupakan klas yan g digunakan untuk membuat file *BufferedImage*.
Atribut
Nama Atribut: *clone* Deskripsi: merupakan atribut yang digunakan untuk mengkloning nilai *BufferedImage* pada *bufferedimage*_0_.
Operasi
Nama Operasi: *createGray*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai abu pada *BufferedImage*.
Nama Operasi: *createRGB*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai RGB pada *BufferedImage*, bertipe *double.*
Nama Operasi: *createRGB*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai RGB pada *BufferedImage*, bertipe *integer*.
Nama Operasi: *create*() Deskripsi: merupaka operasi yang digunakan untuk mendapatkan nilai abu, bertipe *integer*.
Nama Operasi: *create*() Deskripsi: merupaka operasi yang digunakan untuk mendapatkan nilai abu, bertipe *double*.
Nama Operasi : *createBW*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai hitam putih *BufferedImage*.
Nama Operasi: *createPoint* () Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *pointpixel*.
Nama Operasi: *createPointPtr*() Deskripsi:merupakan operasi yang digunakan untuk mendapatkan

nilai *Point* pada *bufferedImage*.

Sumber: Perancangan

g. Klas ImShow

Klas ImShow merupakan klas yang berisikan perintah untuk mengatur ukuran tampilan *interface*.

h. Klas CvTrBI

Klas CvTrBI merupakan klas yang berisikan perintah utnuk mengatur *font* dan mengatur lebar, tinggi *bufferedimage*.

i. Klas PCA. Deskripsi atribut dan operasi dari kelas PCA diperlihatkan pada tabel 4.14 berikut:

Tabel 4.14 Kelas PCA

Nama kelas: PCA
Deskripsi: Kelas PCA merupakan klas yan g digunakan untuk melakukan proses PCA.
Atribut
Nama Atribut: *tData* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *tData*.
Nama Atribut: *tTest* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *tTest*.
Nama Atribut: *meanData* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *meanData*.
Nama Atribut: *aData* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *aData*.

Nama Atribut: lData Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel lData.
Nama Atribut: *eigenFace* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *eigenFace*.
Nama atribut: *eigenVectorSort* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *eigenVectorSort*.
Nama Atribut: *shift* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *shift*.
Nama Atribut: *min* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *min*.
Nama Atribut: *index* Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel *index*.
Nama Atribut: lokasiFile Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel lokasiFile.
Nama Atribut: namaFile Deskripsi : merupakan atribut yang digunakan untuk mendeklarasikan variabel namaFile.
Operasi
Nama Operasi: *Process*() Deskripsi: merupakan operasi yang digunakan untuk memanggil method yang ada di klas PCA.
Nama Operasi: createT() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *gettData*.
Nama Operasi: *calculateMean*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai

setMeanData.
Nama Operasi: *calculateA*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *getaData*.
Nama Operasi: *calculateL*() Deskripsi: merupakan atribut yang digunakan untuk mendapatkan nilai *set1Data*.
Nama Operasi: *calculateEigenFaca*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *eigenface* yang sudah urut.
Nama Operasi: *ProccesTest*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *index*=i.
Nama Operasi: *calculateProjectedImage*() Deskripsi: merupakan operasi yang digunakan untuk mendapatkan nilai *projectedimages*.
Nama Operasi: *calDistance*() Deskripsi: merupaka operasi yang digunakan untuk mendapatkan nilai akar kuadrat nilai *buff*.

Sumber: Perancangan

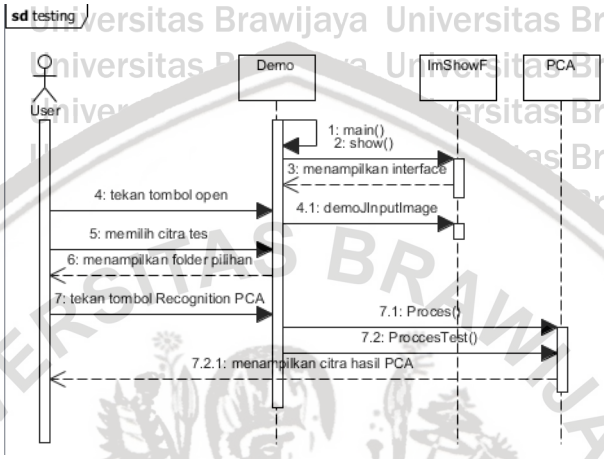
4.2.2 Diagram Sekuensial (*Sequence Diagram*)

Dalam metodologi berorientasi objek untuk menunjukkan bagaimana suatu objek berkomunikasi dengan objek yang lain dengan memperhatikan urutan waktu yang dimodelkan didalam *sequence diagram*. *Sequence diagram* terdiri dari objek-objek yang dipresentasikan dalam pembelajaran didalamnya memuat nama dari objek, pengenalan objek, baca objek yang direpresentasikan dengan garis panah dan *time* (waktu) yang direpresentasikan dengan *vertical progression*. Berikut ini merupakan diagram sekuensial yang ada pada Sistem Pengolahan Citra Digital untuk mendeteksi penyakit dan hama pada tanaman jagung.

4.2.2.1 Diagram *Sequence* Aplikasi *User*

1. Diagram *Sequence* Melakukan Tes

Gambar 4.5merupakan diagram sekuensial Melakukan Testing. Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan proses memilih/memasukkan citra pada sistem.

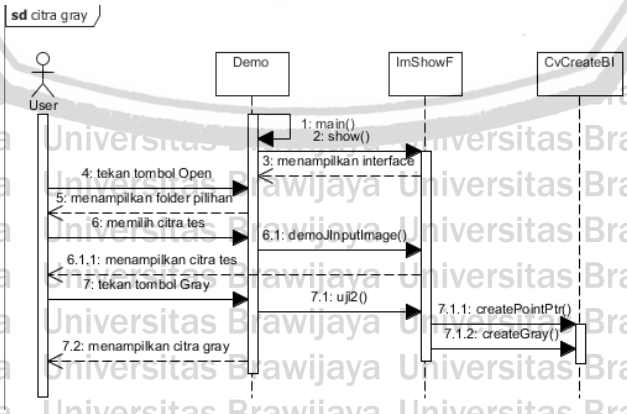


Gambar 4.5 Diagram *sequence* Melakukan Testing

Sumber : Perancangan

2. Diagram *Sequence* Melakukan Tes Citra *Gray*

Gambar 4.6merupakan diagram sekuensial Melakukan Tes Citra *Gray*. Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan proses ubah warna citra *test* menjadi citra *gray*.

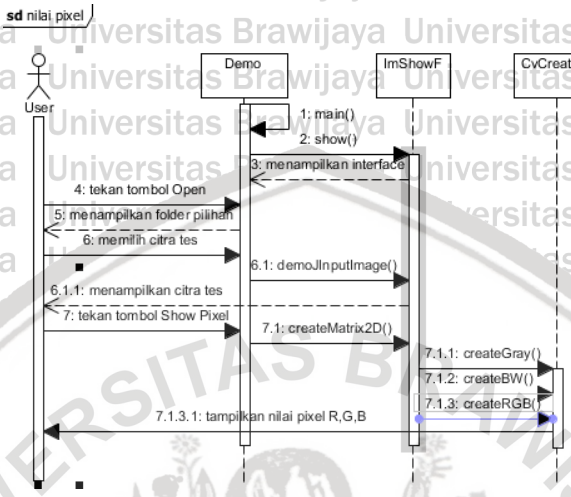


Gambar 4.6 Diagram *sequence* Melakukan Tes Citra *Gray*

Sumber : Perancangan

3. Diagram *Sequence* Menampilkan Nilai *Pixel* Citra

Gambar 4.7 merupakan diagram sekuensial Menampilkan Nilai *Pixel* Citra . Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan proses memilih atau memasukkan citra kemudian menampilkan nilai *pixel* citra.

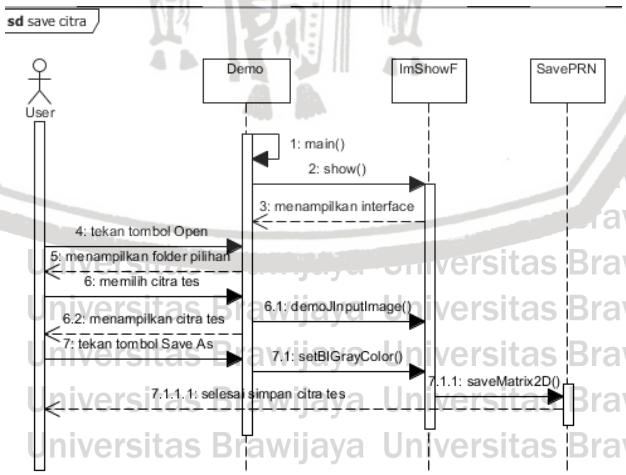


Gambar 4.7 Diagram *sequence* Menampilkan Nilai *Pixel* Citra

Sumber : Perancangan

5 Diagram *Sequence* Menyimpan Citra

Gambar 4.8 merupakan diagram sekuensial Meyimpan Citra. Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan proses menyimpan citra *test.*



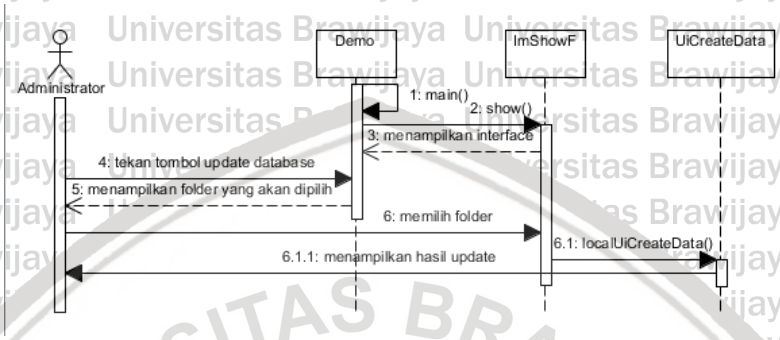
Gambar 4.8 Diagram *sequence* Menyimpan Citra

Sumber : Perancangan

4.2.2.2 Diagram *Sequence* Aplikasi *Administrator*

1. Diagram *Sequence* Memperbarui *Database*

Gambar 4.9 merupakan diagram sekuensial Memperbarui *Database*. Diagram sekuensial ini menggambarkan interaksi ketika *administrator* melakukan proses *update database*.



Gambar 4.9 Diagram *sequence* Memperbarui *Database*

Sumber : Perancangan

4.3 Perancangan Algoritma

Sistem pengolahan citra digital ini menggunakan algoritma *Principal Component Analysis* (PCA). Pada penulisan ini penelitian hanya dicantumkan perancangan algoritma dari beberapa proses (operasi) saja (tidak untuk keseluruhan operasi).

4.3.1 Perancangan Algoritma *Principal Component Analysis*

4.3.1.1 Perancangan Algoritma *Process*

Pada algoritma ini terdapat pemanggilan *methodcreateT* untuk mengeluarkan nilai *Mean*, A (hasil *transpose* Data), L (hasil *methodgettData* dan *getMeanData*), *Matrix* dari *set1Data*, *eigenface* secara urut, dan mencari nilai vektor dari citra.

```
Nama algoritma: Process
Deklarasi:
    • Vector vector
Deskripsi:
    • Input: vector
    • Proses:
      1. Lakukan proses createT(vector);
      2. Lakukan proses calculateMean(gettData());
      3. Lakukan proses calculateA(gettData(), getMeanData());
      4. Lakukan proses calculateL(getaData());
      5. Lakukan proses calculateEigenFaca(getlData(),
         getaData());
      6. Mendefinisikan variabel vector2 pada klas Vector
      7. vector2.add(0, new DataPersist("d:/citra.jpg", "jpg",
         new double[][] { { 3.0D, 5.0D, 3.0D }, { 2.0D, 1.0D,
         8.0D}};
    • Output : mendapatkan nilai vector2 dari klas
    DataPersist.
```

Gambar 4.10 Perancangan Algoritma *Process*

Sumber: Perancangan

Penjelasan algoritma Perhitungan Process() pada Gambar 4.10 yaitu:

1. Baris 1 untuk memanggil proses createT()
2. Baris 2 untuk memanggil proses calculateMean()
3. Baris 3 untuk memanggil proses calculateA()
4. Baris 4 untuk memanggil proses calculateL()
5. Baris 5 untuk memanggil proses calculateEigenFaca()
6. Baris 6 untuk mendefinisikan variabel vector2 = variabel *Vector* baru
7. Baris 7 untuk menambah nilai vector2 dengan memanggil proses di klas *DataPersist*

4.3.1.2 Perancangan Algoritma *createT*

Algoritma ini merupakan proses untuk memanggil proses *createT*. Untuk keterangan algoritma *createT* akan dijelaskan di bawah ini.

```
Nama algoritma: createT
Deklarasi:
    • Vector vector
    • Integer K=0, i=0, j=0, baris
    • Double dataMatrix
Deskripsi:
    • Input: dataPersis, vector
    • Proses:
      1. Memanggil klas DataPersist
      2. Mencari nilai baris
      3. settData(new double[baris][vector.size()]);
      4. for (int k = 0; k < vector.size(); k++) maka
      5. dataPersis = (DataPersist)vector.get(k);
      6. dataMatrix = dataPersis.getData();
      7. for (int i = 0; i < dataMatrix.length; i++) maka
      8. for (int j = 0; j < dataMatrix[0].length; j++) maka
      9. gettData()[K][k] = dataMatrix[i][j];
    • Output : mendapatkan nilai array gettData K,k sama
      dengan nilai array dataMatrix i,j.
```

Gambar 4.11 Perancangan Algoritma *createT*

Sumber: Perancangan

Penjelasan algoritma Perhitungan createT() pada Gambar 4.11 yaitu:

1. Baris 1-2 merupakan perhitungan mendapatkan nilai baris dengan memanggil klas *DataPersist*
2. Baris 3-9 merupakan perhitungan mendapatkan nilai *arraylistgettData* K,k.

4.3.1.3 Perancangan Algoritma *calculateMean*

Proses penghitungan *calculateMean* dilakukan dengan menggunakan nilai Data untuk mencari nilai *buff* kemudian dilakukan proses pada klas *JaMa* lalu proses *transpose*. Hasil akhir yang diperoleh adalah nilai *setMeanData* adalah nilai *summBuff*.

```
Nama algoritma: calculateMean
Deklarasi:
    • Double : arraylist buff, arraylist sumBuff
Deskripsi:
    • Input: buff, sumBuff
    • Proses:
      1. Menghitung nilai buff;
      2. Mencari nilai sumBuff dengan menjumlah hasil JaMa dengan
         nilai buff.
      3. sumBuff = JaMa.divide(sumBuff, buff.length);
      4. setMeanData(sumBuff);
```

Gambar 4.12 Perancangan Algoritma *calculateMean*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateMean() pada Gambar 4.12 yaitu:

1. Baris 1-2 merupakan pendeklarasian variabel buff dan *sumBuff*.
2. Baris 3 merupakan proses untuk mendapatkan nilai *sumBuff*
3. Baris 4 merupakan pendeklarasian bahwa nilai *setMeanData* yaitu nilai *sumBuff*.

4.3.1.4 Perancangan Algoritma *calculateA*

Proses perhitungan digunakan untuk menghitung nilai *getaData* dari panjang nilai *vector* T dengan nilai mean yaitu panjangnya nilai *vector* T ke-x.

```
Nama algoritma: calculateA
Deklarasi:
    • Double : arraylist T, mean
    • Integer : i=0, j=0
Deskripsi:
    • Input: T, mean
    • Proses:
      1. Mendeklarasikan setaData
      2. for (int i = 0; i < T.length; i++) maka
      3. for (int j = 0; j < T[0].length; j++) maka
      4. menghitung nilai getaData()[i][j]
    • Output: didapatkan hasil citra setelah dilakukan proses
      writeEdges.
```

Gambar 4.13 Perancangan Algoritma *calculateA*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateA() pada Gambar 4.13 yaitu:

1. Baris 1 merupakan pemanggilan proses setaData().
2. Baris 2-3 merupakan proses perulangan untuk mendapatkan nilai i dan j.
3. Baris 4 merupakan pemanggilan proses getaData() untuk mendapatkan nilai arraylist i dan j.

4.3.1.5 Perancangan Algoritma *calculateL*

Perancangan algoritma calculateL menggunakan nilai a untuk dimasukkan pada klas Matrix kemudian mendapatkan nilai akhir set1Data.

4.3.1.6 Perancangan Algoritma *calculateEigenFace*

Perancangan algoritma *calculateEigenFace*dimulai dengan mencari nilai *eigenVector* kemudian diurut lalu mencari *eigenFace* dan diurutkan. Untuk lebih jelasnya seperti di bawah ini :

```
Nama algoritma: calculateEigenFace
Deklarasi:
    • EigenvalueDecomposition eig
    • Matrix matrix
    • Integer i2=0, j2=0, j, i
    • Double arraylist eigenVector, eigenValu, a, L
Deskripsi:
    • Input: eig, arraylist eigenVector, arraylist eigenValu
    • Proses:
      1. Mencari nilai eigenVector
      2. Mencari nilai eigenValu
      3. Melakukan setEigenVectorSort = arraylist panjang
         eigenVector dan arraylist panjang eigenVector ke-x;
      4. Menghitung nilai j2 = eigenVector[0].length - 1;
      5. for (int j = 0; j < eigenVector[0].length; j++)maka
      6. for (int i = 0; i < eigenVector.length; i++) maka
      7. getEigenVectorSort()[i][j] = eigenVector[i2][j2];
      8. nilai i2 + 1;
      9. nilai j2 – 1;
      10.   matrix = Matrix(L);
      11.   setEigenFace(matrix.times(new
         Matrix(getEigenVectorSort())).getArrayCopy());
    • Output : menampilkan nilai eigenFace dan eigenVector
      yang sudah diurutkan.
```

Gambar 4.14 Perancangan Algoritma *calculateEigenFace*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateEigenFace() pada Gambar 4.14 yaitu:

1. Baris 1-2 untuk mendeklarasikan variabel *eigenVector* dan *eigenValu*
2. Baris 3 merupakan pemanggilan proses setEigenVectorSort()
3. Baris 4 untuk mendapatkan nilai variabel j2
4. Baris 5-9 merupakan proses perulangan untuk mendapatkan nilai *eigenVector* yang sudah diurutkan.
5. Baris 10 untuk menentukan nilai *matrix* = *Matrix*(L)
6. Baris 11 merupakan pemanggilan proses setEigenFace().

4.3.1.7 Perancangan Algoritma *ProccessTest*

Perancangan algoritma *ProccessTest*dimulai dengan menentukan nilai variabel *DataPersis* kemudian mencari nilai *Delta* , dan seterusnya. Untuk lebih jelasnya seperti di bawah ini :

Nama algoritma: ProccessTest
Deklarasi:

- Data Persist DataPersis
- Matrix matrix
- Integer k=0, i=0, j=0,
- Double arraylist projectedImage, mean, a, eigenface, projectedImageTest, delta
- Vector test

Deskripsi:

- Input: eig, arraylist eigenVector, arraylist eigenValu
- Proses:

```
1. Menghitung nilai dataPersis pada klas DataPersist
2. mencari nilai tTest = dataPersis.getData();
3. projectedImage = calculateProjectedImage(A,
   eigenface);
4. mencari nilai delta = new double[mean.length][1];
5. for (int i = 0; i < this.tTest.length; i++) maka
6. for (int j = 0; j < this.tTest[0].length; j++) maka
7. delta[k][0] = (this.tTest[i][j] - mean[k]);
8. nilai k +1;
9. mencari nilai projectedImageTest
10.   setShift(new double[projectedImage[0].length]);
11.   for (int j = 0; j<projectedImage[0].length;
   j++)maka
12.   getShift()[j] = calDistance(projectedImageTest,
   JaMa.GetColum(projectedImage,j));
13.   this.min = getShift()[0];
14.   this.index = 0.0D;
15.   for (int i = 0; i < getShift().length; i++)maka
16.   if (this.min >= getShift()[i]) maka
17.   this.min = getShift()[i];
18.   this.index = i;
```

- Output: menunjukkan variabel this menunjuk variabel index = i.

Gambar 4.15 Perancangan Algoritma *ProccessTest*

Sumber: Perancangan

Penjelasan algoritma Perhitungan ProccessTest() pada Gambar 4.15 yaitu:

1. Baris 1 untuk mendeklarasikan variabel *dataPersis*
2. Baris 2 untuk menunjukkan bahwa variabel *this* ditunjukkan oleh *tTest*
3. Baris 3 merupakan pemanggilan proses calculateProjectedImage()
4. Baris 4 untuk mendeklarasikan variabel *delta*
5. Baris 5-8 merupakan perulangan untuk mendapatkan nilai *Delta*
6. Baris 9 untuk mendeklarasikan variabel *projectedImageTest* adalah nilai matrix baru yang diproses pada klas *Matrix*.
7. Baris 10 merupakan pemanggilan proses setShift()

8. Baris 11-12 merupakan proses perulangan untuk mendapatkan nilai *getShift* dan memanggil proses getShift(), calDistance()
9. Baris 13 menunjukkan bahwa variabel *this* menunjukkan variabel *min*
10. Baris 14 menunjukkan bahwa variabel *this* menunjukkan variabel *index*
11. Baris 15 merupakan proses perulangan untuk menunjukkan variabel *this* menunjuk variabel *index* = i.

4.3.1.8 Perancangan Algoritma *calculateProjectedImage*

```
Nama algoritma: calculateProjectedImage
Deklarasi:
    • Integer : j=0, i=0
    • Double arraylist projectedImage, An, A, eigenface
Deskripsi:
    • Input: A, eigenface
    • Proses:
        1. Menghitung nilai Matrix eigT
        2. double[][] projectedImage = new
           double[A[0].length][eigenface[0].length];
        3. for (int j = 0; j < A[0].length; j++) maka
        4. double[][] An = eigT.times(new Matrix(JaMa.GetColum(A,
           j))).getArrayCopy();
        5. menghitung nilai array AnSum = JaMa.sum(An);
        6. for (int i = 0; i < An.length; i++) maka
        7. projectedImage[i][j] = An[i][0];
        8. mengembalikan nilai projectedImage;
    • Output: mendapatkan nilai projectedImage.
```

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateProjectedImage() pada Gambar 4.16 yaitu:

1. Baris 1-2 merupakan pedeklarasian variabel *eigt* dan *projectedImage*
2. Baris 3-7 merupakan operasi perulangan untuk mendapatkan nilai *arraylistprojectedImage* i,j.
3. Baris 8 merupakan proses pengembalian nilai *projectedImage*.

4.3.1.9 Perancangan Algoritma calDistance

Nama algoritma: calDistance
Deklarasi:
- Double : buff=0.0D, arraylist a, arraylist b
- Integer i=0

Deskripsi:
- Input: buff, a, b
- Proses:
  1. for (int i = 0; i < a.length; i++) maka
  2. buff += Math.pow(a[i][0] - b[i][0], 2.0D);
  3. mengembalika nilai akar kuadrat buff
- Output: didapatkan hasil citra setelah dilakukan proses writeEdges.

Gambar 4.17 Perancangan Algoritma *calDistance*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calDistance() pada Gambar 4.17 yaitu:

1. Baris 1-2 merupakan proses perulangan untuk mendapatkan nilai *buff*
2. Baris 3 merupakan proses perhitungan mencari akar kuadrat dari nilai *buff*.

4.3.2 Perancangan Algoritma EigenValueDecomposition

4.3.2.1 Perancangan Algoritma *tred2*

Algoritma *tred2* digunakan sebagai algoritma untuk membuat *tridiagonalform*.

Nama algoritma: tred2

Deklarasi:

- Double : scale, h,f,g, hh,h
- Integer j=0, i,k=0

Deskripsi:

- Input: j=0
- Proses:

```
for (int j = 0; j < this.n; j++) maka
this.d[j] = this.V[(this.n - 1)][j];
for (int i = this.n - 1; i > 0; i--)maka
for (int k = 0; k < i; k++) maka
scale += Math.abs(this.d[k]);}
if (scale == 0.0D) maka
this.e[i] = this.d[(i - 1)];
for (int j = 0; j < i; j++) maka
this.d[j] = this.V[(i - 1)][j];
    this.V[i][j] = 0.0D;
    this.V[j][i] = 0.0Dmaka
    for (int k = 0; k < i; k++) maka
    this.d[k] /= scale;
    h += this.d[k] * this.d[k];}
    mencari nilai f = this.d[(i - 1)];
    mencari nilai g = Math.sqrt(h);
    if (f > 0.0D) {
    nilai variabel g = -g;}
    this.e[i] = (scale * g);
    h -= f * g;
    this.d[(i - 1)] = (f - g);
    for (int j = 0; j < i; j++) maka
    this.e[j] = 0.0D;}
    for (int j = 0; j < i; j++) maka
    f = this.d[j];
    this.V[j][i] = f;
    g = this.e[j] + this.V[j][j] * f;
    for (int k = j + 1; k <= i - 1; k++) maka
    g += this.V[k][j] * this.d[k];
    this.e[k] += this.V[k][j] * f;}
    this.e[j] = g;}
    for (int j = 0; j < i; j++) maka
    this.e[j] /= h;
    f += this.e[j] * this.d[j];}
    double hh = f / (h + h);
    for (int j = 0; j < i; j++) maka
    this.e[j] -= hh * this.d[j];}
    for (int j = 0; j < i; j++) maka
    f = this.d[j];
    g = this.e[j];
    for (int k = j; k <= i - 1; k++) maka
    this.V[k][j] -= f * this.e[k] + g * this.d[k];}
    this.d[j] = this.V[(i - 1)][j];
    this.V[i][j] = 0.0D;}}
    this.d[i] = h;}
    for (int i = 0; i < this.n - 1; i++) maka
    this.V[(this.n - 1)][i] = this.V[i][i];
    this.V[i][i] = 1.0D;
    double h = this.d[(i + 1)];
    if (h != 0.0D) maka
    for (int k = 0; k <= i; k++) maka
```

```
52.this.d[k] = (this.V[k][(i + 1)] / h);}
53.for (int j = 0; j <= i; j++) maka
54.for (int k = 0; k <= i; k++) maka
55.g += this.V[k][(i + 1)] * this.V[k][j];}
56.for (int k = 0; k <= i; k++) maka
57.this.V[k][j] -= g * this.d[k];}}}
58.for (int k = 0; k <= i; k++) maka
59.this.V[k][(i + 1)] = 0.0D;}}
60.for (int j = 0; j < this.n; j++) maka
61.this.d[j] = this.V[(this.n - 1)][j];
62.this.V[(this.n - 1)][j] = 0.0D;}
63.this.V[(this.n - 1)][(this.n - 1)] = 1.0D;
64.this.e[0] = 0.0D;}
```

- Output: mendapatkan akumulasi nilai transformasi.

Gambar 4.18 Perancangan Algoritma *tred2*

Sumber: Perancangan

Penjelasan algoritma Perhitungan tred2() pada Gambar 4.18 yaitu:

1. Baris 1-2 merupakan proses perulangan untuk mendapatkan nilai d[j].
2. Baris 3-49 merupakan proses perulangan untuk mengurangi *Housholder* pada *formtridiagonal*.
3. Baris 50-72 merupakan proses perulangan untuk mendapatkan akumulasi transformasi.

4.3.2.2 Perancangan Algoritma *tql2*

Algoritma ini berisi tentang perhitungan *tridiagonal* simetris dengan menggunakan *QLAlgoritma*.

Nama algoritma: tql2
Deklarasi:

- Double : f,tst1,eps,g,p,r,dl1,h,c,c2,c3,e11,s,s2
- Integer: i,l,m,iter,k,j

Deskripsi:

- Input: i=0
- Proses:

```
1. for (int i = 1; i < this.n; i++) {
2. this.e[(i - 1)] = this.e[i];}
3. this.e[(this.n - 1)] = 0.0D;
4. menghitung nilai eps = Math.pow(2.0D, -52.0D);
5. for (int l = 0; l < this.n; l++){
6. mencari nilai tst1
7. while ((m < this.n) && (Math.abs(this.e[m]) > eps
   * tst1))maka
8. nilai m+1
9. if (m > l) maka
10.   do lakukan
11.   iter += 1;
12.   mencari nilai variabel g = this.d[l];
13.   mencari nilai variabel p = (this.d[(l + 1)] -
   g) / (2.0D * this.e[l]);
14.   mencari nilai variabel r = Maths.hypot(p,
   1.0D);
15.   if (p < 0.0D) maka
16.   r = -r;}
17.   this.d[l] = (this.e[l] / (p + r));
18.   this.d[(l + 1)] = (this.e[l] * (p + r));
19.   mencari nilai variabel dl1 = this.d[(l + 1)];
20.   mencari nilai variabel h = g - this.d[l];
21.   for (int i = l + 2; i < this.n; i++) maka
22.   this.d[i] -= h;}
23.   f += h;
24.   p = this.d[m];
25.   nilai variabel c = 1.0D;
26.   nilai variabel c2 = c;
27.   nilai variabel c3 = c;
28.   double el1 = this.e[(l + 1)];
29.   for (int i = m - 1; i >= l; i--)maka
30.   nilai variabel c3 = c2;
31.   nilai variabel c2 = c;
32.   nilai variabel s2 = s;
33.   nilai variabel g = c * this.e[i];
34.   nilai variabel h = c * p;
35.   r = Maths.hypot(p, this.e[i]);
36.   this.e[(i + 1)] = (s * r);
37.   nilai variabel s = this.e[i] / r;
38.   nilai variabel c = p / r;
39.   nilai variabel p = c * this.d[i] - s * g;
40.   this.d[(i + 1)] = (h + s * (c * g + s *
   this.d[i]));
41.   for (int k = 0; k < this.n; k++) maka
42.   nilai variabel h = this.V[k][(i + 1)];
43.   this.V[k][(i + 1)] = (s * this.V[k][i] + c *
   h);
44.   this.V[k][i] = (c * this.V[k][i] - s * h);}}
```

```
45.   nilai variabel p = -s * s2 * c3 * el1 *
   this.e[l] / dl1;
46.   this.e[l] = (s * p);
47.   this.d[l] = (c * p);}
48.   while (Math.abs(this.e[l]) > eps * tst1);maka
49.   this.d[l] += f;
50.   this.e[l] = 0.0D;}
51.   for (int i = 0; i < this.n - 1; i++) maka
52.   variabel k = i;
53.   double p = this.d[i];
54.   for (int j = i + 1; j < this.n; j++) maka
55.   if (this.d[j] < p) maka
56.   variabel k = j;
57.   p = this.d[j];}}
58.   if (k != i) maka
59.   this.d[k] = this.d[i];
60.   this.d[i] = p;
61.   for (int j = 0; j < this.n; j++) maka
62.   p = this.V[j][i];
63.   this.V[j][i] = this.V[j][k];
64.   this.V[j][k] = p;}}}}
```

- Output: mengurutkan nilai eigenvealues dan menyesuaikan nilai vector

Sumber: Perancangan

Penjelasan algoritma Perhitungan tql2() pada Gambar 4.19 yaitu:

1. Baris 1-55 merupakan proses perulangan untuk mendapatkan nilai *convergence*.
2. Baris 56-69 merupakan proses perulangan untuk mengurutkan nilai *eigenvalues* dan menyesuaikan dengan nilai *vector*.

4.3.2.3 Perancangan Algoritma *orthes*

Algoritma ini digunakan untuk mengurangi nonsimetris pada *formHessenberg*.

Nama algoritma: orthes

Deklarasi:

- Double : scale,h,g,f
- Integer:low=0,high,m,i ,j

Deskripsi:

- Input: low=0, high=this.n - 1
- Proses:

```
for (int m = low + 1; m <= high - 1; m++){
double scale = 0.0D;
for (int i = m; i <= high; i++) {
scale += Math.abs(this.H[i][(m - 1)]);}
if (scale != 0.0D){
double h = 0.0D;
for (int i = high; i >= m; i--) {
this.ort[i] = (this.H[i][(m - 1)] / scale);
h += this.ort[i] * this.ort[i];}
  double g = Math.sqrt(h);
  if (this.ort[m] > 0.0D) {
  g = -g;}
  h -= this.ort[m] * g;
  this.ort[m] -= g;
  for (int j = m; j < this.n; j++) {
  double f = 0.0D;
  for (int i = high; i >= m; i--) {
  f += this.ort[i] * this.H[i][j];}
  f /= h;
  for (int i = m; i <= high; i++) {
  this.H[i][j] -= f * this.ort[i];}}
  for (int i = 0; i <= high; i++) {
  double f = 0.0D;
  for (int j = high; j >= m; j--) {
  f += this.ort[j] * this.H[i][j];}
  f /= h;
  for (int j = m; j <= high; j++) {
  this.H[i][j] -= f * this.ort[j];}}
  this.ort[m] = (scale * this.ort[m]);
  this.H[m][(m - 1)] = (scale * g);}}
  for (int i = 0; i < this.n; i++) {
  for (int j = 0; j < this.n; j++) {
  this.V[i][j] = (i == j ? 1.0D : 0.0D);}}
  for (int m = high - 1; m >= low + 1; m--)
  if (this.H[m][(m - 1)] != 0.0D) {
  for (int i = m + 1; i <= high; i++) {
  this.ort[i] = this.H[i][(m - 1)];}
  for (int j = m; j <= high; j++) {
  double g = 0.0D;
  for (int i = m; i <= high; i++) {
  g += this.ort[i] * this.V[i][j];}
  g = g / this.ort[m] / this.H[m][(m - 1)];
  for (int i = m; i <= high; i++)
  this.V[i][j] += g * this.ort[i];}}}
```

- Output: akumulasi nilai transformasi

Gambar 4.20 Perancangan Algoritma *orthes*

Sumber: Perancangan

Penjelasan algoritma Perhitungan orthes() pada Gambar 4.20 yaitu:

1. Baris 1-30 merupakan proses perulangan untuk menghitung transformasi *Householder*
2. Baris 31-44 merupakan operasi perulangan untuk menghitung akumulasi transformasi *Algol'sortran*.

4.3.2.4 Perancangan Algoritma *cdiv*

Perancangan algoritma ini digunakan untuk mencari nilai *scalar division*.

4.3.2.5 Perancangan Algoritma *hqr2*

Perancangan algoritma selanjutnya adalah *hqr2* digunakan untuk mengurangi nilai nonsimetris dari *Hessenberg* ke *formSchur*. Pada algoritma ini terdapat proses mengecek nilai *convergence*, modifikasi kolom dan baris dan proses yang terakhir adalah melakukan transformasi kembali untuk mendapatkan *matrix* awal *eigenvectors*.

4.3.2.6 Perancangan Algoritma *EigenValueDecomposition*

Algoritma *EigenValueDecomposition* digunakan untuk memanggil method yang ada pada klas *EigenvalueDecomposition*.

```
Nama algoritma: EigenValueDecomposition
Deklarasi:
    • Double : arraylist A
    • Integer:j,i
    • Matrix: Arg
Deskripsi:
    • Input: Arg
    • Proses:
      1. this.issymmetric = true;
      2. for (int j = 0; (j < this.n & this.issymmetric); j++)
         maka
      3. for (int i = 0; (i < this.n & this.issymmetric); i++)
         maka
      4. this.issymmetric = (A[i][j] == A[j][i]);}}
      5. if (this.issymmetric) maka
      6. for (int i = 0; i < this.n; i++) maka
      7. for (int j = 0; j < this.n; j++) maka
      8. this.V[i][j] = A[i][j];}}
      9. memanggil method tred2();
      10.   memanggil method tql2() maka
      11.   this.H = new double[this.n][this.n];
      12.   this.ort = new double[this.n];
      13.   for (int j = 0; j < this.n; j++) maka
      14.   for (int i = 0; i < this.n; i++) maka
      15.   this.H[i][j] = A[i][j];}}
      16.   memanggil method orthes();
      17.   memanggil method hqr2();}}
    •     Output:mendapatkan nilai eigen value.
```

Sumber: Perancangan

Penjelasan algoritma Perhitungan EigenValueDecomposition() pada Gambar 4.21 yaitu:

1. Baris 1-4 merupakan proses perulangan untuk mendapatkan nilai *issymmetric*
2. Baris 5-8 merupakan peroses perulangan untuk mendapatkan nilai V.
3. Baris 9 merupakan pemanggilan proses tred2()
4. Baris 10 merupakan pemanggilan proses tql2()
5. Baris 11- 16 merupakan merupakan proses perulangan untuk diagnolisasi.
6. Baris 17 merupakan pemanggilan proses orthes()
7. Baris 18 merupakan pemanggilan proses hqr2()

4.3.3 Perancangan Algoritma *SingularValueDecomposition*

4.3.3.1 Perancangan Algoritma *SingularValueDecomposition*

Perancangan algoritma ***SingularValueDecomposition()*** merupakan algoritma yang digunakan untuk melakukan semua proses perhitungan SVD dan memanggil beberapa *method* yang ada pada klas***SingularValueDecomposition().***

4.3.3.2 Perancangan Algoritma getU

Perancangan algoritma getU() digunakan untuk mengembalikan nilai *vectorsingular* kiri.

4.3.3.3 Perancangan Algoritma getV

Perancangan algoritma getV() digunakan untuk mengembalikan nilai *vectorsingular* kanan.

4.3.3.4 Perancangan Algoritma getSingularValues

Perancangan algoritma getSingularValues() digunakan untuk mengembalikan nilai *array* satu dimensi dari nilai *singular*.

4.3.3.5 Perancangan Algoritma getS

Perancangan algoritma getS() digunakan untuk mengembalikan nilai diagonal matrix dari nilai *singular*.

4.3.3.6 Perancangan Algoritma rank

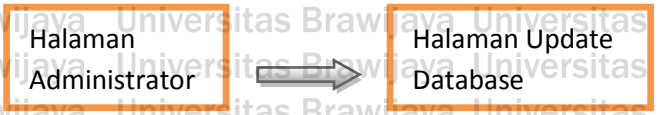
Perancangan algoritma rank() digunakan untuk mencari nilai urutan nomer *matrix* yang efektif.

4.4 Perancangan Antar Muka

Perancangan antar muka dibutuhkan untuk mewakili keadaan sebenarnya dari aplikasi yang akan dibangun. Sistem ini hanya memiliki satu yang halaman yang digunakan untuk mendeteksi citra dan memiliki beberapa menu yang digunakan untuk memperbarui *database* hingga merubah citra menjadi *grayscale*, dan lain-lain. Berikut ini gambaran antarmuka yang ditunjukkan dengan *sitemap* halaman prediksi citra.

4.4.1 Perancangan Antar Muka Halaman *Administrator*

Halaman prediksi citra yang disediakan sistem untuk mengolah data tes dan memperbarui *database*. Pada sistem ini, halaman untuk admin hanya terdiri atas memperbarui *database*. Sedangkan halaman untuk *client* hanya terdiri atas halaman testing. *Site map* halaman *administrator* ditunjukkan pada Gambar 4.22.

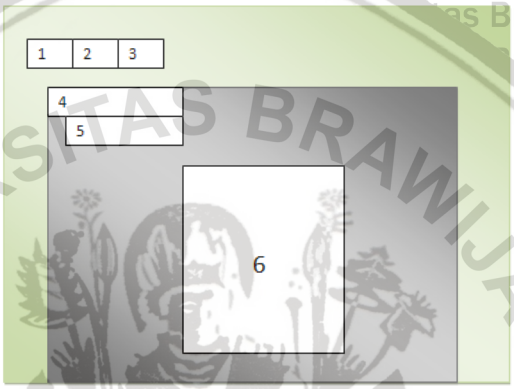


Gambar 4.22*Site Map* Halaman *Administrator*

Sumber: Perancangan

1. **Halaman *UpdateDatabase***

Administrator diahadapkan pada halaman testing yang terdiri atas beberapa menu. pada halaman testing *administrator* memilih menu *Format* lalu *UpdateDatabase*. Perancangan tampilan untuk halaman *updatedatabase* ditunjukkan pada Gambar 4.23.



Gambar 4.23 Perancangan Tampilan Halaman Utama *Administrator*

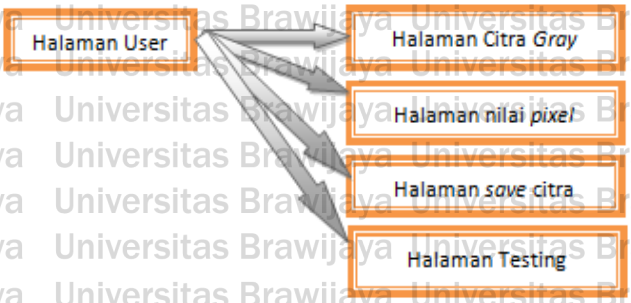
Sumber: Perancangan

Keterangan Gambar 4.23:

1. Tombol untuk memilih menu *File*
2. Tombol untuk memilih menu *Format*
3. Tombol untuk memilih menu *Version*
4. Nama halaman Pilih Direktori
5. Tombol untuk memilih *File* yang akan dilakukan training pada halaman Pilih Direktori
6. Menampilkan gambar yang telah dilakukan *updatedatabase*.

4.4.2 Perancangan Antar Muka Halaman *User*

Site map menu *user* ditunjukkan pada Gambar 4.24.

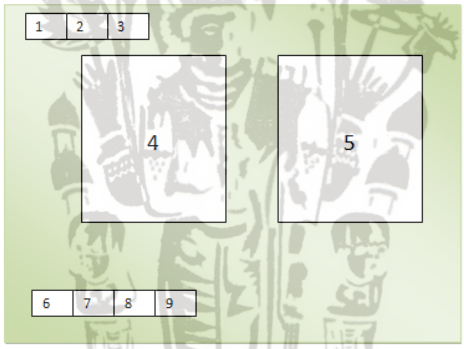


Gambar 4.24*Site Map* Halaman *User*

Sumber: Perancangan

1. Halaman Testing

Halaman testing yang disediakan untuk *client* melakukan testing citra. Pada halaman testing terdapat *form* Citra *Test* dan Citra Hasil. Perancangan tampilan halaman Deteksi Hama ditunjukkan pada Gambar 4.25.



Gambar 4.25 Perancangan Tampilan Halaman Testing

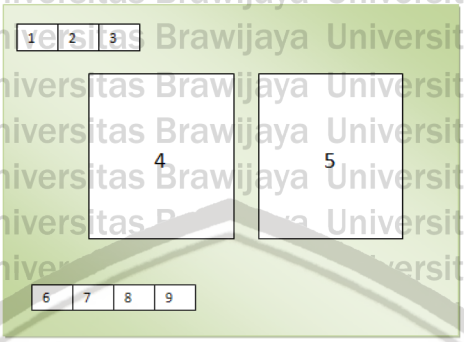
Sumber: Perancangan

Keterangan Gambar 4.25:

1. Tombol untuk memilih menu *File*
2. Tombol untuk memilih menu *Format*
3. Tombol untuk memilih menu *Version*
4. Untuk menampilkan citra *test* yang telah dipilih
5. Untuk menampilkan citra hasil setelah diproses *Recognition* PCA
6. Untuk menampilkan lebar citra *test*
7. Untuk menampilkan tinggi citra *test*
8. Untuk menampilkan format citra yang diproses
9. Untuk menampilkan proses akhir yang dilakukan

2. Halaman Citra *Gray*

Halaman citra gray disediakan untuk user melakukan perubahan citra menjadi citra *gray*. Perancangan tampilan halaman Citra *Gray* ditunjukkan pada Gambar 4.26.



Gambar 4.26 Perancangan Tampilan Halaman Citra *Gray*

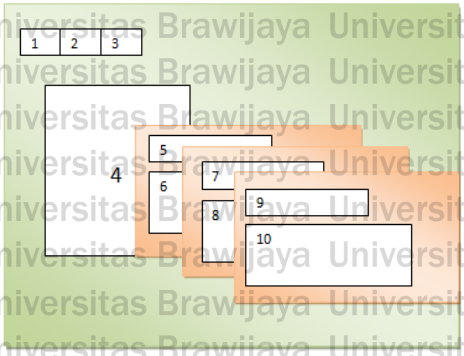
Sumber: Perancangan

Keterangan Gambar 4.26:

1. Tombol untuk memilih menu *File*
2. Tombol untuk memilih menu *Format*
3. Tombol untuk memilih menu *Version*
4. Untuk menampilkan citra *test* yang telah dipilih (jika *user* memilih menu *Gray* maka citra akan langsung berubah di *field* citra tes)
5. Untuk menampilkan citra hasil setelah diproses *Recognition* PCA
6. Untuk menampilkan lebar citra *test*
7. Untuk menampilkan tinggi citra *test*
8. Untuk menampilkan format citra yang diproses
9. Untuk menampilkan proses akhir yang dilakukan

3. Halaman Nilai *Pixel* Citra

Halaman nilai *pixel* citra disediakan untuk user mengetahui nilai *pixel* dari citra tes. Perancangan tampilan halaman nilai *pixel* citra ditunjukkan pada Gambar 4.27.



Gambar 4.27 Perancangan Tampilan Halaman nilai *pixel* citra

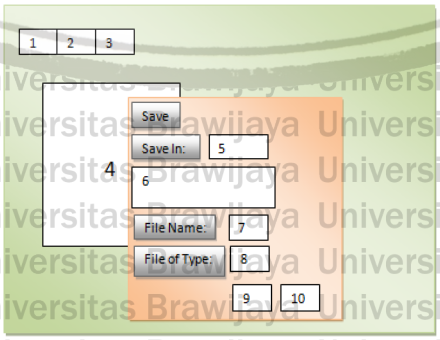
Sumber: Perancangan

Keterangan Gambar 4.27:

1. Tombol untuk memilih menu *File*
2. Tombol untuk memilih menu *Format*
3. Tombol untuk memilih menu *Version*
4. Untuk menampilkan citra *test* yang telah dipilih
5. Nama halaman *red pixel*
6. *Field* untuk nilai pixel merah
7. Nama halaman *green pixel*
8. *Field* untuk nilai *pixel* hijau
9. Nama halaman *blue pixel*
10. *Field* untuk nilai *pixel* biru

4. Halaman *Save* Citra

Halaman *save* citra disediakan untuk *user* agar dapat menyimpan data tes citra sebagai *file*. Perancangan tampilan halaman *save* citra ditunjukkan pada Gambar 4.28.



Gambar 4.28 Perancangan Tampilan Halaman *Save* citra

Sumber: Perancangan

Keterangan Gambar 4.28:

1. Tombol untuk memilih menu *File*
2. Tombol untuk memilih menu *Format*
3. Tombol untuk memilih menu *Version*
4. Untuk menampilkan citra *test* yang telah dipilih
5. *Field* untuk menampilkan folder yang dipilih
6. *Field* untuk menampilkan nama folder yang tersedia
7. *Field* untuk *rename* nama citra
8. *Field* untuk memilih *type* citra yang akan disimpan
9. Tombol *Save*
10. Tombol *Cancel*

BAB V
IMPLEMENTASI

Pada bab ini dibahas mengenai implementasi perangkat lunak berdasarkan hasil yang telah diperoleh dari analisis kebutuhan dan proses perancangan perangkat lunak yang dibuat. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, batasan-batasan dalam implementasi, implementasi tiap klas pada *file* program, dan implementasi algoritma.

5.1 Spesifikasi Sistem

Hasil analisis kebutuhan dan perancangan perangkat lunak yang telah diuraikan pada Bab 4 menjadi acuan untuk melakukan implementasi menjadi sistem yang dapat berfungsi sesuai dengan kebutuhan. Spesifikasi sistem diimplementasikan pada spesifikasi perangkat keras dan perangkat lunak.

5.1.1 Spesifikasi Perangkat Keras

Pengembangan sistem pengolahan citra digital untuk mendeteksi hama dan penyakit tanaman jagung menggunakan sebuah komputer dengan spesifikasi perangkat keras yang dijelaskan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras Komputer

Nama Komponen	**Spesifikasi**
Prosesor	Intel (R) Core(TM) i5 CPU M 520 @2.40GHz 2.40GHz
Memori(RAM)	4 GB
Tipe Sistem	64-bit Operating System
Kartu Grafis	GeForce GT 330M

Sumber: Implementasi

5.1.2 Spesifikasi Perangkat Lunak

Pengembangan sistem pengolahan citra digital untuk mendeteksi hama dan penyakit tanaman jagung menggunakan perangkat lunak dengan spesifikasi yang dijelaskan pada Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Lunak Komputer

Sistem Operasi	**Microsoft Windows 7 Home Premium Service Pack 1**
Bahasa Pemrograman	Java
***Tools* pemrograman**	JDK 1.6.0_02
Lingkungan pemrograman	Java(TM) 2 Runtime Environment, Standard Edition (build 1.6.0_06) , Apache MINA Framework 2.00-M3

IDE (*Integrated Development Environment*)	Netbeans 6.9.1

Sumber: Implementasi

5.2 Batasan-Batasan Implementasi

Beberapa batasan dalam mengimplementasikan sistem adalah sebagai berikut:

- *Input* yang diterima oleh sistem berupa gambar dengan tipe .jpg atau .bmp .
- *Output* yang diterima *user* berupa gambar yang sesuai dengan data yang diuji.
- IDE (*Integrated Development Environment*) yang digunakan adalah Netbeans 6.9.
- Platform pengembangan yang digunakan adalah JSE 6 (*Java Standard Edition* 6).
- Metode yang digunakan yaitu *Pricipal Component Analysis*.
- Hama dan penyakit yang bisa dideteksi hanyalah gambar hama yang bisa terlihat bentuk seperti Lalat Bibit, Ulat Grayak, Penggerek batang, penggerek tongkol, kutu daun, tikus, belalang, kepik, ulat jengkal jagung.
- Sistem ini terbatas hanya untuk memprediksi hasil pengolahan citra digital hama dan penyakit tanaman jagung.

5.3 Implementasi Kelas pada File Program

Setiap kelas yang telah dirancang pada proses perancangan direalisasikan pada sebuah *file* program *.java. Tabel 5.3 menjelaskan mengenai pasangan antara kelas dengan *file* program yang digunakan untuk mengimplementasikannya.

Tabel 5.3 Implementasi kelas pada kode program *.java

No	Paket	Nama Kelas Manager	Nama File Program Manager .java
1.	a	a	a.java
2.	convert	Any2Any	Any2Any.java
3.		Im2BI	Im2BI.java
4.		Im2Bw	Im2Bw.java
5.		Int2logical	Int2logical.java
6.		Int2uint8	Int2uint8.java
7.	cv	CvBIinfo	CvBIinfo.java
8.		CvCreateBI	CvCreateBI.java
9.		CvFilterBI	CvFilterBI.java
10.		CvReadBI	CvReadBI.java
11.		CvSaveBI	CvSaveBI.java
12.		CvTrBI	CvTrBI.java

13.		JImage	Jimage.java
14.	ext	CoG	CoG.java
15.		FeatGlcm	FeatGlcm.java
16.		GLCM	GLCM.java
17.		Labelling	Labelling.java
18.		OrienEllips	OrienEllips.java
19.	ext.pca	DataPersist	DataPersist.java
20.		PCA	PCA.java
21.	jama	CholeskyDecomposition	CholeskyDecomposition.java
22.		EigenvalueDecompositio n	EigenvalueDecomposition.java
23.		LUDecomposition	LUDecomposition.java
24.		Matrix	Matrix.java
25.		QRDecomposition	QRDecomposition.java
26.		SingularValueDecomposi tion	SingularValueDecomposition.java
27.	jama.uti	Maths	Maths.java
28.	math	Conv2D	Conv2D.java
29.		Hist	Hist.java
30.		JaMa	JaMa.java
31.	math.threshold	Otsu	Otsu.java
32.	demo	A	A.java
33.		B	B.java
34.		C	C.java
35.		ImShowF	ImShowF.java
36.		JConfirm	JConfirm.java
37.		JError	Jerror.java
38.		UiCreateData	UiCreateData.java
39.		d	d.java
40.		e	e.java
41.		f	f.java
42.		g	g.java
43.		h	h.java
44.		i	i.java
45.		j	J.java
46.		k	k.java
47.		l	l.java
48.		m	m.java
49.		n	n.java
50.		o	o.java
51.		p	p.java
52.		q	q.java
53.		r	r.java
54.		s	s.java
55.		t	t.java
56.		u	u.java
57.		v	v.java
58.		w	w.java

59.		x	x.java
60.		y	y.java
61.		z	z.java

Sumber: Implementasi

5.4 Implementasi Algoritma

Sistem pengolahan citra digital ini menggunakan algoritma *Principal Component Analysis* untuk melakukan ekstraksi fitur dari citra hama yang akan diuji.

5.4.1 Implementasi Algoritma *Principal Component Analysis*

5.4.1.1 Implementasi Algoritma Perhitungan *Process*

Pada algoritma ini terdapat pemanggilan *method createT* untuk mengeluarkan nilai *Mean*, A (hasil *transpose* Data), L (hasil *method gettData* dan *getMeanData*), *Matrix* dari *set1Data*, *eigenface* secara urut, dan mencari nilai *vector* dari citra.

```
8. public void Process(Vector vector){
9.    System.out.println("T");
10.         createT(vector);
11.         System.out.println("Mean");
12.         calculateMean(gettData());
13.         System.out.println("A");
14.         calculateA(gettData(), getMeanData());
15.         System.out.println("L");
16.         calculateL(getaData());
17.   calculateEigenFaca(getlData(), getaData());
18.   Vector vector2 = new Vector();
19.   vector2.add(0, new DataPersist("d:/citra.jpg", "jpg",
      new double[][] { { 3.0D, 5.0D, 3.0D }, { 2.0D, 1.0D,
      8.0D}})
20.   );
```

Gambar 5.1 Implementasi Algoritma *Process*

Sumber: Perancangan

Penjelasan algoritma Perhitungan Process() pada Gambar 5.1 yaitu:

8. Baris 2 untuk mencetak nilai T
9. Baris 3 untuk memanggil proses createT()
10. Baris 4 untuk mencetak variabel "*Mean*"
11. Baris 5 untuk memanggil proses calculateMean()
12. Baris 6 untuk mencetak variabel A

13. Baris 7 untuk memanggil proses calculateA()
14. Baris 8 untuk mencetak variabel L
15. Baris 9 untuk memanggil proses calculateL()
16. Baris 10 untuk memanggil proses calculateEigenFaca()
17. Baris 11 untuk mendefinisikan variabel *vector2* = variabel *Vector* baru
18. Baris 12 untuk menambah nilai vector2 dengan memanggil proses di klas *DataPersist*

5.4.1.2 Implementasi Algoritma *createT*

Algoritma ini merupakan proses untuk memanggil proses *createT*. Untuk keterangan algoritma *createT* akan dijelaskan di bawah ini.

```
1. private void createT(Vector vector)
2. {
3.   DataPersist dataPersis = (DataPersist)vector.get(0);
4.   int baris = dataPersis.getData().length*
   dataPersis.getData()[0].length;
5.   double[][] dataMatrix = (double[][])null;
6.   settData(new double[baris][vector.size()]);
7.   int K = 0;
8.   for (int k = 0; k < vector.size(); k++) {
9.        dataPersis = (DataPersist)vector.get(k);
10.       dataMatrix = dataPersis.getData();
11.       for (int i = 0; i < dataMatrix.length; i++){
12.         for (int j = 0; j < dataMatrix[0].length; j++){
13.         gettData()[K][k] = dataMatrix[i][j];
14.         K++;}}
15.  K = 0;}}
```

Gambar 5.2 Implementasi Algoritma *createT*

Sumber: Perancangan

Penjelasan algoritma Perhitungan createT() pada Gambar 5.2 yaitu:

3. Baris 3-5 untuk mendeklarasikan variabel *dataPersis* baris, dan *dataMatrix* yang digunakan.
4. Baris 6 untuk memanggil *method settData*
5. Baris 7 untuk mendeklarasikan variabel K=0

6. Baris 8-15 untuk mendapatkan nilai *arraylist gettData* K,k.

5.4.1.3 Implementasi Algoritma *calculateMean*

Proses penghitungan *calculateMean* dilakukan dengan menggunakan nilai Data untuk mencari nilai *buff* kemudian dilakukan proses pada klas *JaMa* lalu proses *transpose*. Hasil akhir yang diperoleh adalah nilai *setMeanData* adalah nilai *summBuff*.

```
1. private void calculateMean(double[][] Data)
2. {
3.       double[][] buff = JaMa.Transpose(Data);
4.       double[] sumBuff = JaMa.sum(buff);
5.       sumBuff = JaMa.divide(sumBuff, buff.length);
6.       setMeanData(sumBuff);
7. }
```

Gambar 5.3 Implementasi Algoritma *calculateMean*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateMean() pada Gambar 5.3 yaitu:

4. Baris 3-4 merupakan pendeklarasian variabel *buff* dan *sumBuff*.
5. Baris 5 merupakan proses untuk mendapatkan nilai *sumBuff*
6. Baris 6 merupakan pendeklarasian bahwa nilai *setMeanData* yaitu nilai *sumBuff*.

5.4.1.4 Implementasi Algoritma *calculateA*

Proses perhitungan digunakan untuk menghitung nilai *getaData* dari panjang nilai *vector* T dengan nilai *mean* yaitu panjangnya nilai *vector* T ke-x.

```
1. private void calculateA(double[][] T, double[] mean) {
2.       setaData(new double[T.length][T[0].length]);
3.        for (int i = 0; i < T.length; i++){
4.           for (int j = 0; j < T[0].length; j++)
5.            getaData()[i][j] = (T[i][j] - mean[i]);
6. }}
```

Gambar 5.4 Implementasi Algoritma *calculateA*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateA() pada Gambar 5.4 yaitu:

4. Baris 2 merupakan pemanggilan proses setaData().

5. Baris 3-4 merupakan proses perulangan untuk mendapatkan nilai i dan j.
6. Baris 5 merupakan pemanggilan proses getaData() untuk mendapatkan nilai *arraylist* i dan j.

5.4.1.5 Implementasi Algoritma *calculateL*

Perancangan algoritma calculateL( ) menggunakan nilai a untuk dimasukkan pada klas *Matrix* kemudian mendapatkan nilai akhir *set1Data*.

5.4.1.6 Implementasi Algoritma *calculateEigenFace*

Perancangan algoritma *calculateEigenFace* dimulai dengan mencari nilai *eigenVector* kemudian diurut lalu mencari *eigenFace* dan diurutkan. Untuk lebih jelasnya seperti di bawah ini :

```
1. private void calculateEigenFaca(double[][]a,double[][]
   L){
2. EigenvalueDecomposition eig = new
   EigenvalueDecomposition(new
   Matrix((double[][])a.clone()));
3.   double[][] eigenVector = eig.getV().getArrayCopy();
4.   double[][] eigenValu = eig.getD().getArrayCopy();
5.   System.out.println("vector");
6.   setEigenVectorSort(new
   double[eigenVector.length][eigenVector[0].length]);
7.   int i2 = 0;
8.   int j2 = eigenVector[0].length - 1;
9.   for (int j = 0; j < eigenVector[0].length; j++){
10.      for (int i = 0; i < eigenVector.length; i++) {
11.      getEigenVectorSort()[i][j] = eigenVector[i2][j2];
12.      i2++;}
13.      i2 = 0;j2--;}
14.      System.out.println("vector urut ");
15.      Matrix matrix = new Matrix(L);
16.      setEigenFace(matrix.times(new
17.      Matrix(getEigenVectorSort()))).getArrayCopy());
18.      System.out.println("eigenFace urut ");
19.      }
```

Gambar 5.5 Implementasi Algoritma *calculateEigenFace*

Sumber: Perancangan

Penjelasan algoritma Perhitungan calculateEigenFace() pada Gambar 5.5 yaitu:

7. Baris 2 untuk mendefinisikan bahwa *eig* adalah variabel baru *EigenvalueDecomposition*
8. Baris 3-4 untuk mendeklarasikan variabel *eigenVector* dan *eigenValu*

9. Baris 5 untuk mencetak nilai *vector*
10. Baris 6 merupakan pemanggilan proses setEigenVectorSort()
11. Baris 7 untuk mendapatkan nilai variabel i2
12. Baris 8 untuk mencari nilai j2
13. Baris 9-19 merupakan proses perulangan untuk mendapatkan nilai *eigenVector* dan *eigenFace* yang sudah diurutkan

5.4.1.7 Implementasi Algoritma *ProccessTest*

Perancangan algoritma *ProccessTest* dimulai dengan menentukan nilai variabel *DataPersis* kemudian mencari nilai *Delta* , dan seterusnya.

```
1.  public void ProccesTest(double[] mean, double[][] A,
    double[][] eigenface, Vector test){
2.    DataPersist dataPersis = (DataPersist)test.get(0);
3.    this.tTest = dataPersis.getData();
4.    double[][] projectedImage = calculateProjectedImage(A,
    eigenface);
5.    double[][] delta = new double[mean.length][1];
6.    System.out.println("Mean ");
7.    int k = 0;
8.    for (int i = 0; i < this.tTest.length; i++){
9.      for (int j = 0; j < this.tTest[0].length; j++){
10.        delta[k][0] = (this.tTest[i][j] - mean[k]);
11.        k++;}}
12.    System.out.println("Delta ");
13.    double[][] projectedImageTest = new Matrix(new
       Matrix(eigenface).transpose().getArray()).times(new
       Matrix(delta)).getArrayCopy();
14.    System.out.println("projectedtestimage ");
15.    setShift(new double[projectedImage[0].length]);
16.    for (int j = 0; j < projectedImage[0].length; j++){
17.       getShift()[j] = calDistance(projectedImageTest,
          JaMa.GetColum(projectedImage, j));}
18.    this.min = getShift()[0];
19.    this.index = 0.0D;
20.    for (int i = 0; i < getShift().length; i++)
21.       if (this.min >= getShift()[i]) {
22.       this.min = getShift()[i];
23.       this.index = i;}}
```

Gambar 5.6 Implementasi Algoritma *ProccessTest*

Sumber: Perancangan

Penjelasan algoritma Perhitungan ProccessTest() pada Gambar 5.6 yaitu:

1. Baris 2 untuk mendeklarasikan variabel *dataPersis*

2. Baris 3 untuk menunjukkan bahwa variabel *this* ditunjukkan oleh *tTest*
3. Baris 4 untuk pemanggilan proses calculateProjectedImage() untuk mencari nilai *projectedImage*
4. Baris 5 untuk mendeklarasikan variabel *delta*
5. Baris 6 untuk mencetak nilai *Mean*
6. Baris 7 untuk mendeklarasikan variabel k=0
7. Baris 8-12 merupakan perulangan untuk mendapatkan nilai *Delta*
8. Baris 13 untuk mencetak nilai *Delta*
9. Baris 14 untuk mencari nilai *projectedImageTest*
10. Baris 15 untuk mencetak nilai *projectedtestimage*
11. Baris 16 untuk memanggil *method* setShift()
12. Baris 17-19 untuk proses perulangan untuk mendapatkan nilai *getShift* dan memanggil proses getShift(), calDistance()
13. Baris 20-21 untuk mendefinisikan variabel *this* menunjukkan variabel min dan variabel *index*
14. Baris 22-26 untuk proses perulangan untuk menunjukkan variabel *this* menunjuk variabel *index* = i.

5.4.1.8 Implementasi Algoritma *calculateProjectedImage*

```
1. private double[][] calculateProjectedImage(double[][] A,
   double[][] eigenface){
2.    Matrix eigT = new Matrix(new
      Matrix(eigenface).transpose().getArrayCopy());
3.    double[][] projectedImage = new
      double[A[0].length][eigenface[0].length];
4.   System.out.println("eigenface tranpose");
5.   for (int j = 0; j < A[0].length; j++) {
6.      double[][] An = eigT.times(new
        Matrix(JaMa.GetColum(A, j))).getArrayCopy();
7.      double[] AnSum = JaMa.sum(An);
8.      for (int i = 0; i < An.length; i++) {
9.          projectedImage[i][j] = An[i][0];}}
10. System.out.println("projected images");
11. return projectedImage;}
```

Gambar 5.7 Implementasi Algoritma *calculateProjectedImage*

Sumber: Perancangan

Penjelasan algoritma perhitungan calculateProjectedImage() pada Gambar 5.7 yaitu:

4. Baris 2-3 merupakan pedeklarasian variabel *eigT* dan *projectedImage*
5. Baris 4 untuk mencetak nilai *eigenface transpose*

6. Baris 5-10 merupakan operasi perulangan untuk mendapatkan nilai *arraylist projectedImage* i,j
7. Baris 11 untuk mencetak nilai *projected images*
8. Baris 12 merupakan proses pengembalian nilai *projectedImage*

5.4.1.9 Implementasi Algoritma calDistance

```
1. private double calDistance(double[][] a, double[][] b)
2. {
3.    double buff = 0.0D;
4.    for (int i = 0; i < a.length; i++) {
5.       buff += Math.pow(a[i][0] - b[i][0], 2.0D);
6.    }
7. return Math.sqrt(buff);
8. }
```

Gambar 5.8 Implememtasi Algoritma *calDistance*

Sumber: Perancangan

Penjelasan algoritma perhitungan calDistance() pada Gambar 5.8 yaitu:

1. Baris 3 untuk mendefinikan variabel *buff* = 0.0D
2. Baris 4-6 merupakan proses perulangan untuk mendapatkan nilai *buff*
3. Baris 7 untuk proses perhitungan mencari akar kuadrat dari nilai *buff*.

5.4.2 Implementasi Algoritma *EigenValueDecomposition*

5.4.2.1 Implementasi Algoritma *tred2*

Algoritma *tred2* digunakan sebagai algoritma untuk membuat *tridiagonal form*.

```
1.private void tred2(){
2.  for (int j = 0; j < this.n; j++) {
3.  this.d[j] = this.V[(this.n - 1)][j];}
4.  for (int i = this.n - 1; i > 0; i--)
5.  double scale = 0.0D;
6.  double h = 0.0D;
7.  for (int k = 0; k < i; k++) {
8.  scale += Math.abs(this.d[k]);}
9.  if (scale == 0.0D) {
10. this.e[i] = this.d[(i - 1)];
11. for (int j = 0; j < i; j++) {
12. this.d[j] = this.V[(i - 1)][j];
13. this.V[i][j] = 0.0D;
14. this.V[j][i] = 0.0D;
15. }}else{
```

```
16.     for (int k = 0; k < i; k++) {
17.       this.d[k] /= scale;
18.       h += this.d[k] * this.d[k];}
19.     double f = this.d[(i - 1)];
20.     double g = Math.sqrt(h);
21.     if (f > 0.0D) {
22.       g = -g;}
23.     this.e[i] = (scale * g);
24.     h -= f * g;
25.     this.d[(i - 1)] = (f - g);
26.     for (int j = 0; j < i; j++) {
27.       this.e[j] = 0.0D;}
28.     for (int j = 0; j < i; j++) {
29.       f = this.d[j];
30.       this.V[j][i] = f;
31.       g = this.e[j] + this.V[j][j] * f;
32.       for (int k = j + 1; k <= i - 1; k++) {
33.         g += this.V[k][j] * this.d[k];
34.         this.e[k] += this.V[k][j] * f;
35.       }this.e[j] = g;}
36.     f = 0.0D;
37.     for (int j = 0; j < i; j++) {
38.       this.e[j] /= h;
39.       f += this.e[j] * this.d[j];}
40.     double hh = f / (h + h);
41.     for (int j = 0; j < i; j++) {
42.       this.e[j] -= hh * this.d[j];
43.     }
44.     for (int j = 0; j < i; j++) {
45.       f = this.d[j];
46.       g = this.e[j];
47.     for (int k = j; k <= i - 1; k++) {
48.           this.V[k][j] -= f * this.e[k] + g *
        this.d[k];
49.     }
50.     this.d[j] = this.V[(i - 1)][j];
51.     this.V[i][j] = 0.0D;
52.     }}
53.     this.d[i] = h;
54.    }
55.             for (int i = 0; i < this.n - 1;
            i++) {
56.               this.V[(this.n - 1)][i] =
            this.V[i][i];
57.               this.V[i][i] = 1.0D;
58.               double h = this.d[(i + 1)];
59.                 if (h != 0.0D) {
60.                   for (int k = 0; k <= i; k++)
            {
61.                     this.d[k] = (this.V[k][(i +
            1)] / h);}
62.                   for (int j = 0; j <= i; j++) {
63.                     double g = 0.0D;
64.                     for (int k = 0; k <= i; k++)
            {
65.                       g += this.V[k][(i + 1)] *
            this.V[k][j];}
```

```
72.    for (int j = 0; j < this.n; j++) {
73.       this.d[j] = this.V[(this.n - 1)][j];
74.       this.V[(this.n - 1)][j] = 0.0D;
75.    }
76.    this.V[(this.n - 1)][(this.n - 1)] = 1.0D;
77.    this.e[0] = 0.0D;
78. }
```

Gambar 5.9 Implementasi Algoritma *tred2*

Sumber: Perancangan

Penjelasan algoritma perhitungan tred2() pada Gambar 5.9 yaitu:

4. Baris 2-3 untuk proses pengulangan nilai d[j].
5. Baris 4-54 untuk mereduksi *householder* ke *form tridiagonal*
6. Baris 16-27 untuk *generate vector householder*
7. Baris 28-54 untuk menerapkan transformasi similiarity ke kolom yang tersedia
8. Baris 55-78 untuk akumulasi transformasi.

5.4.2.2 Implementasi Algoritma *tql2*

Algoritma ini berisi tentang perhitungan tridiagonal simetris dengan menggunakan *QLAlgoritma*.

```
1.    private void tql2(){
2.    for (int i = 1; i < this.n; i++) {
3.    this.e[(i - 1)] = this.e[i];}
4.    this.e[(this.n - 1)] = 0.0D;
5.    double f = 0.0D;
6.    double tst1 = 0.0D;
7.    double eps = Math.pow(2.0D, -52.0D);
8.    for (int l = 0; l < this.n; l++){
9.    tst1 = Math.max(tst1, Math.abs(this.d[l]) +
      Math.abs(this.e[l]));
10.   int m = l;
11.   while ((m < this.n) &&
12.     (Math.abs(this.e[m]) > eps * tst1)){
13.   m++;}
14.   if (m > l) {
15.   int iter = 0;
16.   do {
17.   iter += 1;
18.   double g = this.d[l];
19.   double p = (this.d[(l + 1)] - g) / (2.0D * this.e[l]);
20.   double r = Maths.hypot(p, 1.0D);
21.   if (p < 0.0D) {
22.   r = -r;}
```

```
23.   this.d[l] = (this.e[l] / (p + r));
24.   this.d[(l + 1)] = (this.e[l] * (p + r));
25.   double dl1 = this.d[(l + 1)];
26.   double h = g - this.d[l];
27.   for (int i = l + 2; i < this.n; i++) {
28.   this.d[i] -= h;}
29.   f += h;
30.   p = this.d[m];
31.   double c = 1.0D;
32.   double c2 = c;
33.   double c3 = c;
34.   double el1 = this.e[(l + 1)];
35.   double s = 0.0D;
36.   double s2 = 0.0D;
37.   for (int i = m - 1; i >= l; i--) {
38.   c3 = c2;
39.   c2 = c;
40.   s2 = s;
41.   g = c * this.e[i];
42.   h = c * p;
43.   r = Maths.hypot(p, this.e[i]);
44.   this.e[(i + 1)] = (s * r);
45.   s = this.e[i] / r;
46.   c = p / r;
47.   p = c * this.d[i] - s * g;
48.   this.d[(i + 1)] = (h + s * (c * g + s *
    this.d[i]));
49.   for (int k = 0; k < this.n; k++) {
50.   h = this.V[k][(i + 1)];
51.   this.V[k][(i + 1)] = (s * this.V[k][i] + c *
    h);
52.   this.V[k][i] = (c * this.V[k][i] - s * h);}}
53.   p = -s * s2 * c3 * el1 * this.e[l] / dl1;
54.   this.e[l] = (s * p);
55.   this.d[l] = (c * p);}
56.   while (Math.abs(this.e[l]) > eps * tst1);}
57.   this.d[l] += f;
58.   this.e[l] = 0.0D;}
59.   for (int i = 0; i < this.n - 1; i++) {
60.   int k = i;
61.   double p = this.d[i];
62.   for (int j = i + 1; j < this.n; j++) {
63.   if (this.d[j] < p) {
64.   k = j;
65.   p = this.d[j];}}
66.   if (k != i) {
67.   this.d[k] = this.d[i];
68.   this.d[i] = p;
69.   for (int j = 0; j < this.n; j++) {
70.   p = this.V[j][i];
71.   this.V[j][i] = this.V[j][k];
72.   this.V[j][k] = p;
73.   }}}}
```

Gambar 5.10 Implementasi Algoritma *tql2*

Sumber: Perancangan

Penjelasan algoritma perhitungan tql2() pada Gambar 5.10 yaitu:

3. Baris 2-7 untuk mendeklarasikan variabel yang akan digunakan
4. Baris 8-13 untuk mencari elemen terkecil subdiagonal
5. Baris 14-15 untuk menentukan d[l] adalah *eigenvalue* jika m==1 jika tidak dilakukan iterasi
6. Baris 18-29 untuk menghitung shift secara lengkap (*implicit shift*)
7. Baris 30-48 untuk mengaplikasi QL transformasi
8. Baris 49-55 untuk akumulasi transformasi
9. Baris 56-58 untuk mengecek *convergence*
10. Baris 59-73 untuk mengurutkan eigenvalues dan menyesuaikan dengan *eigenvectors*.

5.4.2.3 Implementasi Algoritma *orthes*

Algoritma ini digunakan untuk mengurangi nonsimetris pada *form Hessenberg*.

```
1. private void orthes(){
2.    int low = 0;
3.    int high = this.n - 1;
4.    for (int m = low + 1; m <= high - 1; m++){
5.      double scale = 0.0D;
6.       for (int i = m; i <= high; i++) {
7.        scale += Math.abs(this.H[i][(m - 1)]);}
8.    if (scale != 0.0D){
9.     double h = 0.0D;
10.    for (int i = high; i >= m; i--) {
11.     this.ort[i] = (this.H[i][(m - 1)] / scale);
12.     h += this.ort[i] * this.ort[i];}
13.    double g = Math.sqrt(h);
14.     if (this.ort[m] > 0.0D) {
15.     g = -g;}
16.    h -= this.ort[m] * g;
17.    this.ort[m] -= g;
18.    for (int j = m; j < this.n; j++) {
19.     double f = 0.0D;
20.      for (int i = high; i >= m; i--) {
21.      f += this.ort[i] * this.H[i][j];}
22.  f /= h;
23.     for (int i = m; i <= high; i++) {
24.      this.H[i][j] -= f * this.ort[i];}}
25.     for (int i = 0; i <= high; i++) {
```

```
26.       double f = 0.0D;
27.         for (int j = high; j >= m; j--) {
28.           f += this.ort[j] * this.H[i][j];}
29.         f /= h;
30.         for (int j = m; j <= high; j++) {
31.       this.H[i][j] -= f * this.ort[j];}}
32.       this.ort[m] = (scale * this.ort[m]);
33.      this.H[m][(m - 1)] = (scale * g);}}
34.      for (int i = 0; i < this.n; i++) {
35.        for (int j = 0; j < this.n; j++) {
36.      this.V[i][j] = (i == j ? 1.0D : 0.0D);}}
37.      for (int m = high - 1; m >= low + 1; m--)
38.      if (this.H[m][(m - 1)] != 0.0D) {
39.        for (int i = m + 1; i <= high; i++) {
40.         this.ort[i] = this.H[i][(m - 1)];}
41.        for (int j = m; j <= high; j++) {
42.        double g = 0.0D;
43.          for (int i = m; i <= high; i++) {
44.           g += this.ort[i] * this.V[i][j];}
45.        g = g / this.ort[m] / this.H[m][(m - 1)];
46.        for (int i = m; i <= high; i++)
47.        this.V[i][j] += g * this.ort[i];}}}
```

Gambar 5.11 Implementasi Algoritma *orthes*

Sumber: Perancangan

Penjelasan algoritma perhitungan orthes() pada Gambar 5.11 yaitu:

15. Baris 2-3 untuk mendeklarasikan variabel yang digunakan
16. Baris 4-8 untuk menentukan skala kolom
17. Baris 9-17 untuk menghitung transformasi *Householder*
18. Baris 18-33 untuk mengaplikasikan transformasi *Householder similiarity*
19. Baris 34-44 untuk akumulasi transformasi (*Algol's ortran*)
20. Baris 45-48 untuk nilai g dibagi menjadi 2 untuk menghindari *underflow*

5.4.2.4 Implementasi Algoritma *cdiv*

Perancangan algoritma ini digunakan untuk mencari nilai *scalar division.*

5.4.2.5 Implementasi Algoritma *hqr2*

Perancangan algoritma selanjutnya adalah *hqr2* digunakan untuk mengurangi nilai nonsimetris. Perancangan algoritma selanjutnya adalah *hqr2* digunakan untuk mengurangi nilai nonsimetris dari *Hessenberg* ke *form Schur*. Pada algoritma ini terdapat proses

mengecek nilai *convergence*, modifikasi kolom dan baris dan proses yang terakhir adalah melakukan transformasi kembali untuk mendapatkan *matrix* awal *eigenvectors.*

5.4.2.6 Implementasi Algoritma *EigenValueDecomposition*

Algoritma *EigenValueDecomposition* digunakan untuk memanggil *method* yang ada pada klas *EigenvalueDecomposition.*

```
1. public EigenvalueDecomposition(Matrix Arg){
2.    double[][] A = Arg.getArray();
3.    this.n = Arg.getColumnDimension();
4.    this.V = new double[this.n][this.n];
5.    this.d = new double[this.n];
6.    this.e = new double[this.n];
7.    this.issymmetric = true;
8.    for (int j = 0; (j < this.n & this.issymmetric); j++) {
9.     for (int i = 0; (i < this.n & this.issymmetric); i++){
10.     this.issymmetric = (A[i][j] == A[j][i]);}}
11.   if (this.issymmetric) {
12.    for (int i = 0; i < this.n; i++) {
13.      for (int j = 0; j < this.n; j++) {
14.      this.V[i][j] = A[i][j];}}
15.   tred2();
16.   tql2();
17.   }else {
18.     this.H = new double[this.n][this.n];
19.     this.ort = new double[this.n];
20.      for (int j = 0; j < this.n; j++) {
21.        for (int i = 0; i < this.n; i++) {
22.        this.H[i][j] = A[i][j];}}
23.   orthes();
24.   hqr2();}}
```

Gambar 5.12 Implementasi Algoritma *EigenValueDecomposition*

Sumber: Perancangan

Penjelasan algoritma perhitungan EigenValueDecomposition() pada Gambar 5.12 yaitu:

8. Baris 2-7 untuk mendeklarasikan variabel yang akan digunakan
9. Baris 8-11 untuk mendapatkan nilai *issymmetric*
10. Baris 12-16 untuk mendapatkan nilai *array list* V (i,j)
11. Baris 17 untuk memanggil *method* tred2()
12. Baris 18 untuk memanggil *method* tql2()
13. Baris 19-25 untuk mendapatkan nilai *array list* H(i,j)
14. Baris 26 untuk memanggil *method* orthes()
15. Baris 27 untuk memanggil *method* hqr2()

5.4.3 Implementasi Algoritma *SingularValueDecomposition*

5.4.3.1 Implementasi Algoritma *SingularValueDecomposition*

Perancangan algoritma *SingularValueDecomposition()* merupakan algoritma yang digunakan untuk melakukan semua proses perhitungan SVD dan memanggil beberapa method yang ada pada klas *SingularValueDecomposition().*

5.4.3.2 Implementasi Algoritma getU

Perancangan algoritma getU() digunakan untuk mengembalikan nilai *vector singular* kiri.

5.4.3.3 Implementasi Algoritma getV

Perancangan algoritma getV() digunakan untuk mengembalikan nilai *vector singular* kanan.

5.4.3.4 Implementasi Algoritma getSingularValues

Perancangan algoritma getSingularValues() digunakan untuk mengembalikan nilai *array* satu dimensi dari nilai singular.

5.4.3.5 Implementasi Algoritma getS

Perancangan algoritma getS() digunakan untuk mengembalikan nilai diagonal *matrix* dari nilai singular.

5.4.3.6 Implementasi Algoritma rank

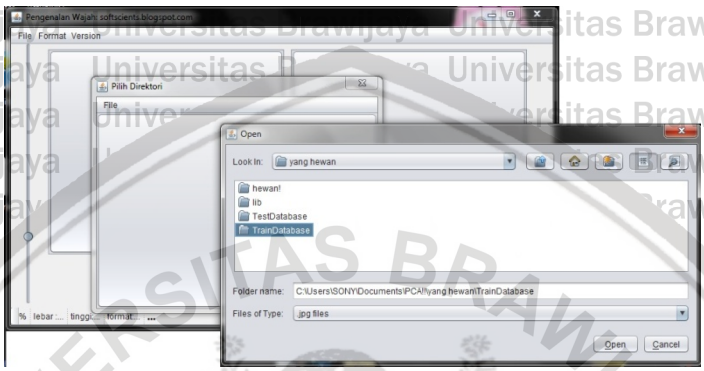
Perancangan algoritma rank() digunakan untuk mencari nilai urutan nomer *matrix* yang efektif.

5.5 Implementasi Antar Muka

5.5.1 Implementasi Halaman Antar Muka Aplikasi Administrasi

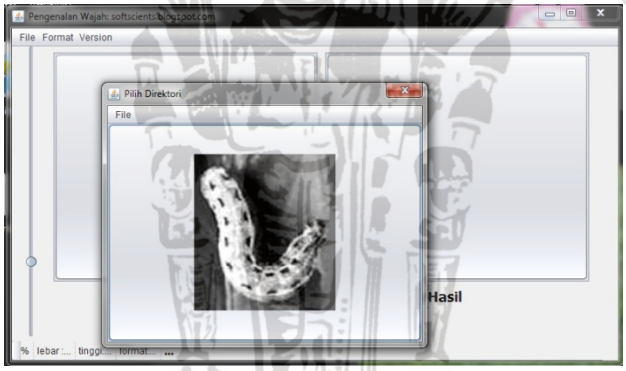
1. Tampilan Halaman Upadate *Database*

Gambar 5.14 dan gambar 5.13 merupakan tampilan dari halaman *Update Database* yang dilakukan oleh *administrator*.



Gambar 5.13 Tampilan Halaman *Update Database* 1

Sumber: Implementasi



Gambar 5.14 Tampilan Halaman *Update Database* 2

Sumber: Implementasi

Pada halaman *Update Database administrator* dapat melakukan *update* data *training* yang sebelumnya sudah dimasukkan pada *folder TrainDatabase*. *Administrator* harus melakukan *update* terhadap *folder* yang sama pada saat mengisi data pada *folder TrainDatabase*.

5.5.2 Implementasi Halaman Antar Muka Aplikasi *User*

1. Tampilan Halaman *Recognition* PCA

Pada halaman *Recognition* PCA user dapat melakukan testing data *test* dengan proses PCA. Alur proses yang dilakukan meliputi *user* harus memilih citra yang akan diuji kemudian memilih *sub-menu Recognition* PCA lalu akan diproses *konvert,load database* dan yang terakhir proses perhitungan jarak. Hasil *output* yang didapat oleh *user* adalah data citra yang sama sesuai dengan citra *test* yang dipilih dengan pose yang hampir sama.

Gambar 5.15 Tampilan Halaman *Recognition* PCA

Sumber: Implementasi

2. Tampilan Halaman *Gray*

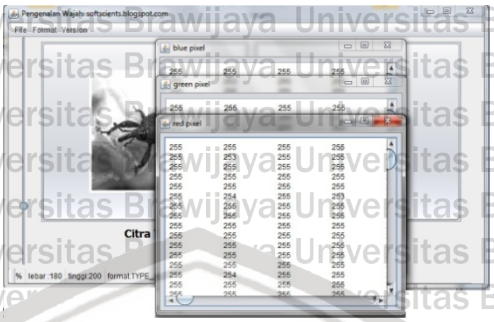
Pada halaman ini *user* dapat memasukkan citra yang akan diubah warna menjadi citra *gray*.

Gambar 5.16 Tampilan Halaman *Gray*

Sumber: Implementasi

3. Tampilan Halaman Nilai Pixel

Pada halaman ini user dapat melihat nilai *pixel* dari *red*, *green*, *blue* untuk citra yang akan diuji dengan memilih *sub-menu show Pixel* dari menu *Format*.

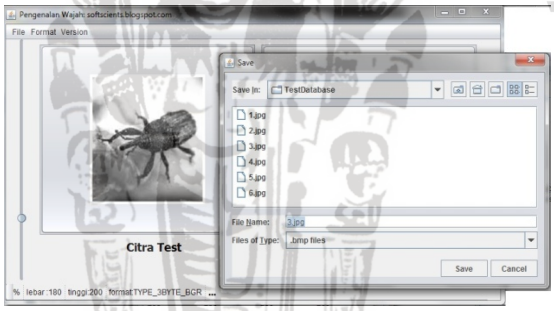


Gambar 5.18 Tampilan Halaman Nilai *Pixel*

Sumber: Implementasi

4. Tampilan Halaman *Save*

Pada halaman ini *user* dapat menyimpan citra yang diuji pada *folder* yang diinginkan dengan memilih *sub-menu Save* pada menu *File*.



Gambar 5.19 Tampilan Halaman *Save*

Sumber: Implementasi

BAB VI
PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan proses pengujian dan analisis terhadap Sistem Pengolahan Citra Digital Untuk Mendeteksi Penyakit dan Hama Pada Tanaman Jagung. Proses pengujian dilakukan melalui tiga tahapan, yaitu pengujian unit, pengujian integrasi, dan pengujian validasi. Pada pengujian unit dan pengujian integrasi akan digunakan teknik pengujian *White Box*. Penulisan pengujian ini mengacu pada penulisan di bab 5. Pada pengujian validasi akan digunakan teknik pengujian *Black Box*.

6.1 Pengujian

Proses pengujian dilakukan melalui dua tahapan yaitu pengujian unit, pengujian integrasi dan pengujian validasi.

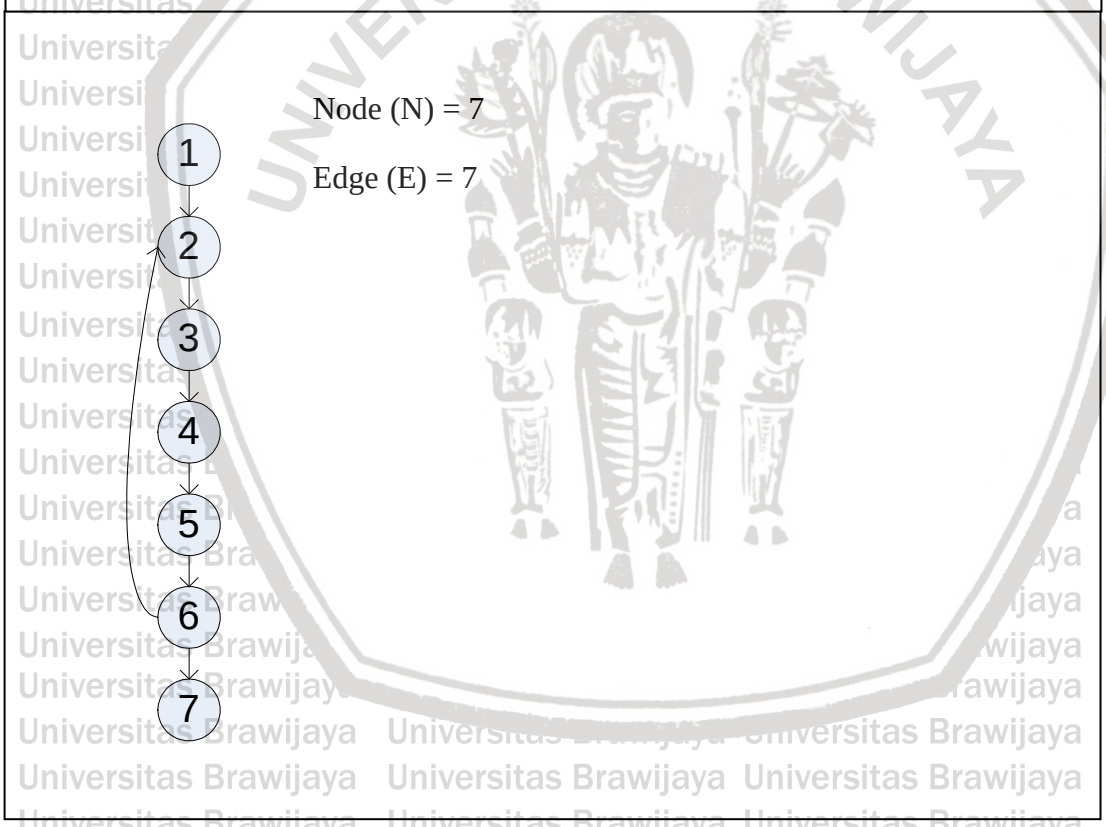
6.1.1 Pengujian Unit

Pada pengujian unit Sistem Pengolahan Citra Digital Untuk Mendeteksi Penyakit dan Hama Pada Tanaman Jagung digunakan teknik pengujian *White Box* dengan teknik *Basis Path Testing*. Pada teknik *Basis Path Testing*, proses pengujian dilakukan dengan memodelkan algoritma pada suatu *flow graph*, menentukan jumlah kompleksitas siklomatis (*cyclomatic complexity*), menentukan sebuah basis set dari jalur independen dan memberikan kasus uji (*test case*) pada setiap basis set yang telah ditentukan. Penulisan laporan penelitian ini hanya dicantumkan hasil pengujian unit untuk algoritma dari beberapa metode (operasi) saja (tidak untuk keseluruhan metode).

6.1.1.1 Pengujian Unit untuk Operasi createT() pada Kelas PCA

Operasi **createT()** merupakan operasi yang digunakan untuk mencari nilai arraylist gettData variabel K, k. Gambar 6.1 menunjukkan proses pemodelan dalam *flow graph* pada operasi **createT()**.

```
private void createT(Vector vector)
{
    DataPersist dataPersis = (DataPersist)vector.get(0);
    int baris = dataPersis.getData().length* dataPersis.getData()[0].length;
    double[][] dataMatrix = (double[][])null;
    settData(new double[baris][vector.size()]);
    int K = 0;
    for (int k = 0; k < vector.size(); k++) {
        dataPersis = (DataPersist)vector.get(k);
        dataMatrix = dataPersis.getData();
        for (int i = 0; i < dataMatrix.length; i++){
            for (int j = 0; j < dataMatrix[0].length; j++){
                gettData()[K][k] = dataMatrix[i][j];
                K++;
            }
        }
        K = 0;
    }
}
```



Gambar 6.1 Pengujian unit untuk algoritma createT()

Sumber: Pengujian

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi createT() menghasilkan jumlah kompleksitas siklomatis (*cyclometic complexity*) melalui persamaan

V(G) = E – N + 2, dimanna V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi garis (garis penghubung antar *node*), dan N merupakan jumlah simpul (*node*).

V(G) = E – N + 2

= 7– 7 + 2

= 2

Dari nilai *cyclometic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur *independent* yaitu:

Jalur 1 : 1 – 2 – 3 – 4 – 5 – 6 – 2 – ...

Jalur 2: 1 – 2 – 3 – 4 – 5 – 6 – 7

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.1.

Tabel 6.1 *Test Case* untuk Pengujian Unit Operasi createT()

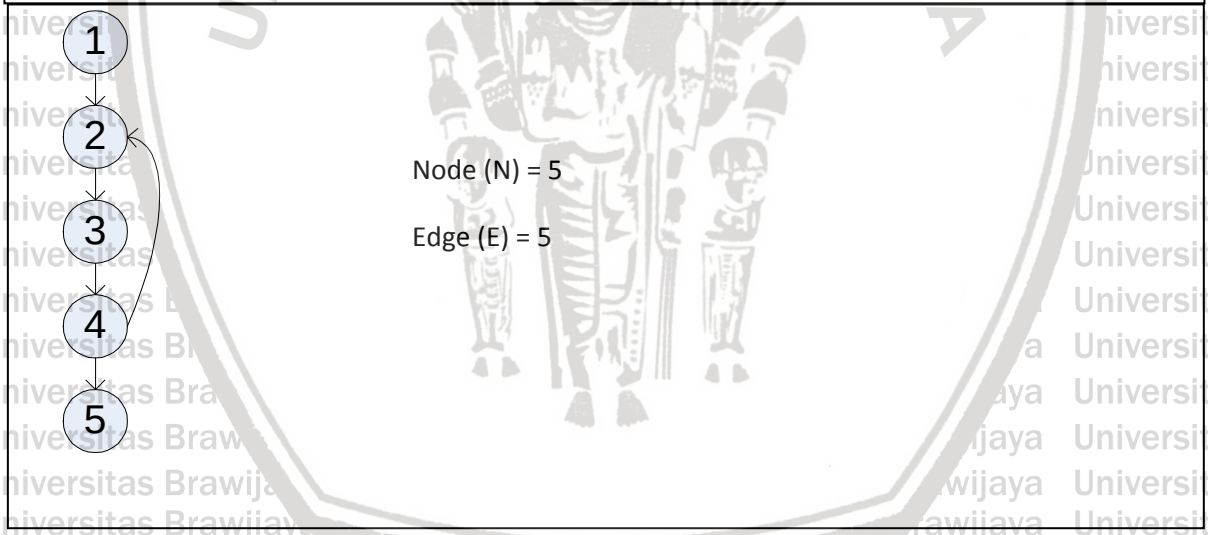
Sumber: Pengujian

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Jika nilai arraylist gettData untuk K != 0	Didapatkan nilai arraylist gettData K, k.	Masih mengulang proses perulangan sampai nilai arraylist gettData K=0, k
2	Jika nilai arraylist gettData untuk K=0	Didapatkan nilai arraylist gettData K, k.	Didapatkan nilai arraylist gettData K, k.

6.1.1.2 Pengujian Unit untuk Operasi setBIGrayColor() pada Kelas CvReadBI

Operasi setBIGrayColor() merupakan operasi yang digunakan untuk merubah RGB citra menjadi citra gray. Gambar 6.2 menunjukkan proses pemodelan dalam *flow graph* pada operasi setBIGrayColor().

```
public void setBIGrayColor(BufferedImage bufferedimage)
{
this.source = new BufferedImage(bufferedimage.getWidth(), bufferedimage.getHeight(),
bufferedimage.getType());
    this.formatType = bufferedimage.getType();
    Raster raster;
    int i = (raster = bufferedimage.getData()).getWidth();
    int i_3_ = raster.getHeight();
    this.grayInt = new int[i][i_3_];
    this.width = i;
    this.height = i_3_;
    raster.getNumBands();
    for (int i_4_ = 0; i_4_ < i; i_4_++){
      for (int i_5_ = 0; i_5_ < i_3_; i_5_++) {
        this.source.setRGB(i_4_, i_5_, bufferedimage.getRGB(i_4_, i_5_));
        this.grayInt[i_4_][i_5_] = raster.getSample(i_4_, i_5_, 0);
      }
    }
  }
```



Gambar 6.2 Pengujian unit untuk algoritma **setBIGrayColor()**

Sumber: Pengujian

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi setBIGrayColor**()** menghasilkan jumlah kompleksitas siklomatis (*cyclometic complexity*) melalui persamaan V(G) = E – N + 2, dimanna V(G) merupakan jumlah kompleksitas

siklomatis, E merupakan sisi garis (garis penghubung antar *node*), dan N merupakan jumlah simpul (*node*).

$$V(G) = E - N + 2$$
$$= 5 - 5 + 2$$
$$= 2$$

Dari nilai *cyclometic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan lima buah basis set dari jalur *independent* yaitu:

Jalur 1 : 1 – 2 – 3 – 4 – 2 – ...

Jalur 2 : 1 – 2 – 3 – 4 – 5

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.2

Tabel 6.2 *Test Case* untuk Pengujian Unit Operasi setBIGrayColor**()**

Sumber: Pengujian

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Terdapat proses perulangan untuk mencari nilai variabel i_4_ dan i_5_ sebagai input grayInt	Mendapatkan nilai raster untuk variabel input pada proses grayInt	Mendapatkan nilai grayInt
2	Tidak ada proses perulangan untuk mencari nilai variabel i_4_ dan i_5_ sebagai input grayInt	Tidak mendapatkan nilai raster untuk variabel input pada proses grayInt	Tidak mendapatkan nilai grayInt

6.1.1.3 Analisis Hasil Pengujian Unit

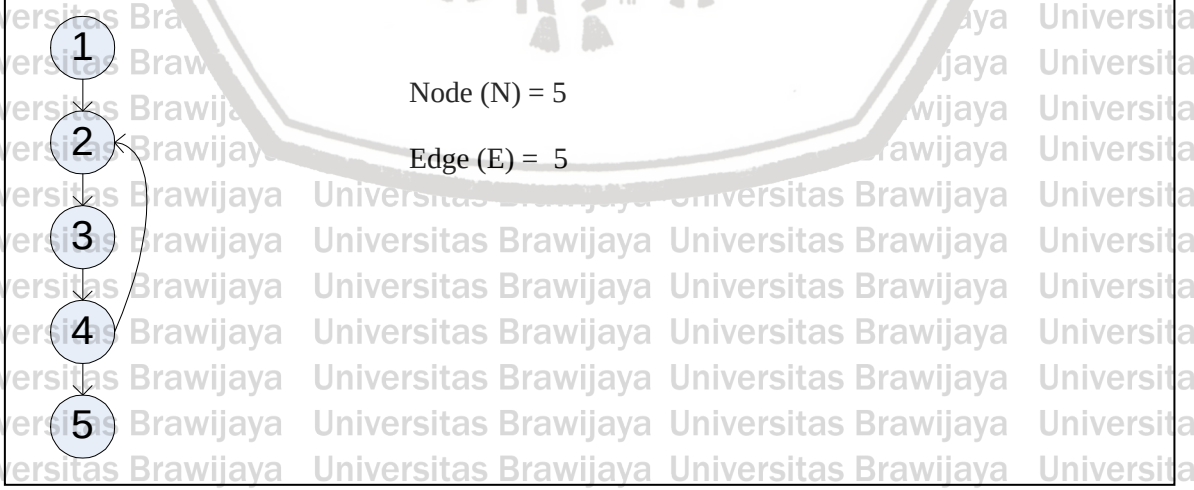
Proses analisis terhadap hasil pengujian unit dilakukan dengan melihat kesesuaian fungsi dari implementasi unit modul yang diuji dengan hasil perancangan perangkat lunak yang telah dirancang sebelumnya. Berdasarkan hal tersebut, maka dapat diambil kesimpulan bahwa unit modul dari program sudah memenuhi kebutuhan fungsional yang telah dirancang pada tahap perancangan.

6.1.2 Pengujian Integrasi

Pengujian integrasi diterapkan pada proses yang mengintegrasikan fungsionalitas dari beberapa kelas untuk melakukan sebuah operasi tertentu. Pada pengujian integrasi yang dijadikan sebagai obyek uji adalah klas-klas yang menggabungkan kinerja dari kelas-kelas yang lain. Pengujian integrasi yang diterapkan dalam sistem ini menggunakan strategi *bottom-up*.

6.1.2.1 Pengujian Integrasi untuk Operasi calculateEigenFaca()pada kelas PCA.

```
private void calculateEigenFaca(double[][]a,double[][] L){
    EigenvalueDecomposition eig = new EigenvalueDecomposition(new
    Matrix((double[][])a.clone()));
    double[][] eigenVector = eig.getV().getArrayCopy();
    double[][] eigenValu = eig.getD().getArrayCopy();
    System.out.println("vector");
    setEigenVectorSort(new double[eigenVector.length][eigenVector[0].length]);
    int i2 = 0;
    int j2 = eigenVector[0].length - 1;
    for (int j = 0; j < eigenVector[0].length; j++){
        for (int i = 0; i < eigenVector.length; i++) {
        getEigenVectorSort()[i][j] = eigenVector[i2][j2];
        i2++;
        }
        i2 = 0;j2--;
    }
    System.out.println("vector urut ");
    Matrix matrix = new Matrix(L);
    setEigenFace(matrix.times(newMatrix(getEigenVectorSort())).getArrayCopy());
    System.out.println("eigenFace urut ");
}
```



Gambar 6.3 Pengujian integrasi untuk algoritma **calculateEigenFaca()**

Sumber: Pengujian

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi hitung1() menghasilkan jumlah kompleksitas siklomatis (*cyclometic complexity*) melalui persamaan V(G) = E – N + 2, dimanna V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi garis (garis penghubung antar *node*), dan N merupakan jumlah simpul (*node*).

V(G) = E – N + 2

= 5 – 5 + 2 = 2

Dari nilai *cyclometic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan lima buah basis set dari jalur *independent* yaitu:

Jalur 1 : 1 – 2 – 3 – 4 – 2 – ...

Jalur 2 : 1 – 2 – 3 – 4 – 5

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.3

Tabel 6.3 *Test Case* untuk Pengujian Integrasi Operasi calculateEigenFaca()

Sumber: Pengujian

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Jika tidak ada proses perulangan variabel i dan j untuk mengurutkan EigenVector.	Mendapatkan nilai EigenVector dan EigenFace yang telah diurutkan	Nilai EigenVector dan EigenFace tidak dapat diurutkan
2	Jika proses semua dilakukan	Mendapatkan nilai EigenVector dan EigenFace yang telah diurutkan	Mendapatkan nilai EigenVector dan EigenFace yang telah diurutkan

6.1.2.2 Pengujian Integrasi untuk Operasi ProccesTest()pada kelas PCA.

```
public void ProccesTest(double[] mean, double[][] A, double[][]
eigenface, Vector test){
  DataPersist dataPersis = (DataPersist)test.get(0);
  this.tTest = dataPersis.getData();
  double[][] projectedImage =
  calculateProjectedImage(A,eigenface);
  double[][] delta = new double[mean.length][1];
  System.out.println("Mean ");
  int k = 0;
  for (int i = 0; i < this.tTest.length; i++){
    for (int j = 0; j < this.tTest[0].length; j++){
    delta[k][0] = (this.tTest[i][j] - mean[k]);
    k++;
    }
  }
  System.out.println("Delta ");
  double[][] projectedImageTest = new Matrix(new
  Matrix(eigenface).transpose().getArray()).times(new
  Matrix(delta)).getArrayCopy();
  System.out.println("projectedtestimage ");
  setShift(new double[projectedImage[0].length]);
  for (int j = 0; j < projectedImage[0].length; j++){
   getShift()[j] =
   calDistance(projectedImageTest,JaMa.GetColum(projectedImage,j))
   ;
  }
  this.min = getShift()[0];
  this.index = 0.0D;
  for (int i = 0; i < getShift().length; i++){
    if (this.min >= getShift()[i]) {
       this.min = getShift()[i];
       this.index = i;
    }
  }
}
```

1 2 3 4 5 6 7 8 9 10 11 12

Gambar 6.4 Pemodelan operasi `ProccesTest()` ke dalam *flow graph*
Sumber: Pengujian

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi ProccesTest() menghasilkan jumlah kompleksitas siklomatis (*cyclometic complexity*) melalui persamaan V(G) = E – N + 2, dimanna V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi garis (garis penghubung antar *node*), dan N merupakan jumlah simpul (*node*).

V(G) = E – N + 2

= 13– 12 + 2 = 3

Dari nilai *cyclometic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan lima buah basis set dari jalur *independent* yaitu:

Jalur 1 : 1 – 2 – 3 – 4 – 3 – 5 – 2 – 6 – 7 – 6 – ...

Jalur 2 : 1 – 2 – 3 – 4 – 5 – 6 – 7 – 6 – 7 – 8 – 9 – 10 – 11 – 9 – ...

Jalur 3 : 1 – 2 – 3 – 5 – 2 – 6 – 7 – 8 – 9 – 10 – 11 – 12

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.4

Tabel 6.4 *Test Case* untuk Pengujian Integrasi Operasi ProccesTest()

Sumber: Pengujian

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Jika terdapat proses perulangan untuk mendapatkan nilai getShift	Mendapatkan nilai getShift dengan memanggil proses calDistance.	Mendapatkan nilai getShift dengan memanggil proses calDistance.
2	Jika terdapat proses perulangan untuk mencari nilai index=i	Mendapatkan nilai index=i	Mendapatkan nilai index=i
3	Jika tidak terdapat proses perulangan nilai j > projectedImages dan proses perulangan untuk mendapatkan nilai getShift	Mendapatkan nilai getShift dengan memanggil proses calDistance	Tidak mendapatkan nilai getShift dengan memanggil proses calDistance

6.1.2.3 Analisis Hasil Pengujian Integrasi

Proses analisis terhadap hasil pengujian dilakukan dengan melihat kesesuaian beberapa unit model yang ,menyusun satu blok fungsi dalam perangkat lunak pengolahan citra digital hama tanaman jagung. berdasarkan hal tersebut, maka dapat diambil kesimpulan bahwa integrasi beberapa unit modul dari program sudah emmenuhi kebutuhan fungsional yang telah dirancang pada tahap perancangan.

6.1.3 Pengujian Validasi

Pengujian validasi digunakan untuk mengetahui apakah sistem yang dibangun sudah benar sesuai dengan yang dibutuhkan. Item-item yang telah dirumuskan dalam daftar kebutuhan dan merupakan hasil analisis kebutuhan akan menjadi acuan untuk melakukan pengujian validasi. Pengujian validasi menggunakan metode pengujian *Black Box,* karena tidak memerlukan untuk berkonsentrasi terhadap alur jalannya algoritma program dan lebih ditekankan untuk menemukan konformitas antara kinerja sistem dengan daftar kebutuhan.

6.1.3.1 Kasus Uji Validasi

a. Kasus Uji Melakukan Testing Sebagai *User*

Nama Kasus Uji : Kasus Uji Melakukan Testing

Objek Uji : Kebutuhan Fungsional Melakukan Tes (SRS_002_01)

Tujuan Pengujian :Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melakukan tes pada sistem sehingga *user* dapat melihat *image* hasil prediksi.

Prosedur Uji : Memilih dan memasukkan gambar yang akan diuji

Hasil yang Diharapkan : Aplikasi dapat menampilkan citra hasil uji.

b. Kasus Uji Melakukan Tes Citra *Gray* Sebagai *User*

Nama Kasus Uji : Kasus Uji Melakukan Tes Citra *Gray*

Objek Uji : Kebutuhan Fungsional Tes Citra *Gray* (SRS_002_02)

Tujuan Pengujian : Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melakukan tes citra *gray*.

Prosedur Uji : Memasukkan citra yang akan diuji kemudian memilih sub-menu *Gray*.

Hasil yang Diharapkan: Aplikasi dapat menampilkan citra hasil *gray*.

c. Kasus Uji Menampilkan Nilai *Pixel* Citra Sebagai *User*

Nama Kasus Uji : Kasus Uji Menampilkan Nilai *Pixel* Citra

Objek Uji : Kebutuhan Fungsional Menampilkan Nilai *Pixel* Citra (SRS_002_03)

Tujuan Pengujian : Pengujian dilakukan untuk memastikan bahwa aplikasi

dapat memenuhi kebutuhan fungsional untuk menampilkan nilai *pixel* citra

Prosedur Uji : Memasukkan citra yang akan diuji kemudian memilih sub-menu *Show Pixel*

Hasil yang Diharapkan: Aplikasi dapat menampilkan nilai pixel citra

d. Kasus Uji Menyimpan Citra Sebagai *User*

Nama Kasus Uji : Kasus Uji Menyimpan Citra

Objek Uji : Kebutuhan Fungsional Menyimpan Citra (SRS_002_04)

Tujuan Pengujian : Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk menyimpan citra.

Prosedur Uji : Memasukkan data citra digital, kemudian memilih sub-menu "*Save*"

Hasil yang Diharapkan: Aplikasi dapat melakukan penyimpanan citra pada folder yang dipilih.

e. Kasus Uji Memperbarui *Database* Citra Sebagai *Administrator*

Nama Kasus Uji : Kasus Uji Memperbarui *Database* Citra

Objek Uji : Kebutuhan Fungsional Memperbarui *Database* Citra (SRS_001_01)

Tujuan Pengujian : Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memperbarui *database* citra.

Prosedur Uji : Memilih sub-menu "*UpdateDatabase*" lalu memilih

folder yang berisi data *training* citrakemudian menekan tombol "*Open*."

Hasil yang Diharapkan: Aplikasi dapat melakukan *update database* data training citra.

6.1.3.2 Hasil Pengujian Validasi

Dari kasus uji yang telah dilaksanakan sesuai dengan prosedur pengujian pada sub pokok bahasan 6.1.3.1, didapatkan hasil seperti ditunjukkan pada Tabel 6.5.

Tabel 6.5 *Test Case* untuk Pengujian Validasi

No.	Kasus Uji	Hasil yang Didapatkan	Status
1.	Melakukan Testing Sebagai *User*	Aplikasi dapat menampilkan citra hasil uji.	Valid
2.	Melakukan Tes Citra *Gray* Sebagai *User*	Aplikasi dapat menampilkan citra hasil *gray*.	Valid
3.	Menyimpan Citra Sebagai *User*	Aplikasi dapat melakukan penyimpanan citra pada folder yang dipilih	Valid
4.	Memperbarui *Database* Citra Sebagai *Administrator*	Aplikasi dapat melakukan *update database* data training citra	Valid

Sumber: Pengujian

6.1.3.3 Analisis Hasil Pengujian Validasi

Proses analisis terhadap hasil pengujian validasi dilakukan dengan melihat konformitas antara hasil kinerja sistem dengan daftar kebutuhan. Berdasarkan hasil pengujian validasi dapat disimpulkan bahwa implementasi dan fungsionalitas perangkat lunak pengolahan citra digital hama tanaman jagung telah memenuhi kebutuhan yang telah dijabarkan pada tahap analisis kebutuhan.

6.2 Pengujian Validitas Sistem

Untuk menguji validitas dari sistem, maka dilakukan uji validitas sistem terhadap data pelatihan yang telah ada. Sistem dikatakan memiliki kinerja tinggi apabila output yang dihasilkan oleh sistem pengolahan citra digital untuk hama dan penyakit jagung ini memiliki output yang sama dengan output yang ada pada database.

6.2.1 Pengujian Kualitas Citra Digital

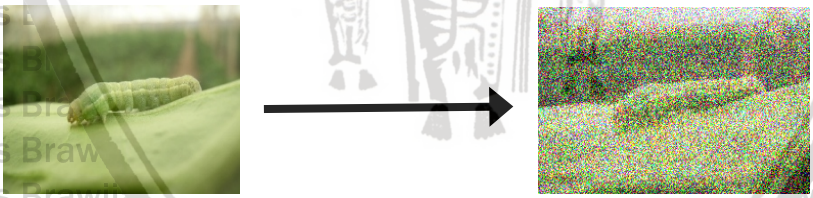
Pada sistem pengolahan citra digital ini menggunakan input citra hama tanaman jagung yang kemudian dilakukan proses PCA untuk mengolah citra tersebut. Pada gambar 6.5 akan diperlihatkan histogram warna dari citra awal hama tanaman jagung.

Gambar 6.5 Gambar citra awal dan histogram citra hama jagung

Sumber : Pengujian

Pengujian kualitas citra digital dilakukan dengan pengujian identifikasi citra digital dengan pola citra digital yang diberi noise dengan prosentase tertentu. Pemberian noise dilakukan dengan menggunakan software pengolah gambar Adobe Photoshop Elements 8.0 dengan tipe *uniform noise*. Pengujian kualitas citra digital ini bertujuan untuk mengetahui kemampuan perangkat lunak Pengolahan Citra Digital Untuk Mendeteksi Hama dan Penyakit Jagung dalam melakukan prediksi citra digital dengan pola citra digital yang tingkat ketajamannya berbeda-beda.

Gambar 6.6 menunjukkan gradasi *noise* sebanyak 60% pada pengujian kualitas citra digital hama jagung. Pengujian kualitas citra digital ditunjukkan pada Tabel 6.6.



Gambar 6.6 Gradasi *noise* sebanyak 60% pada pengujian kualitas citra digital hama jagung

Sumber : Pengujian

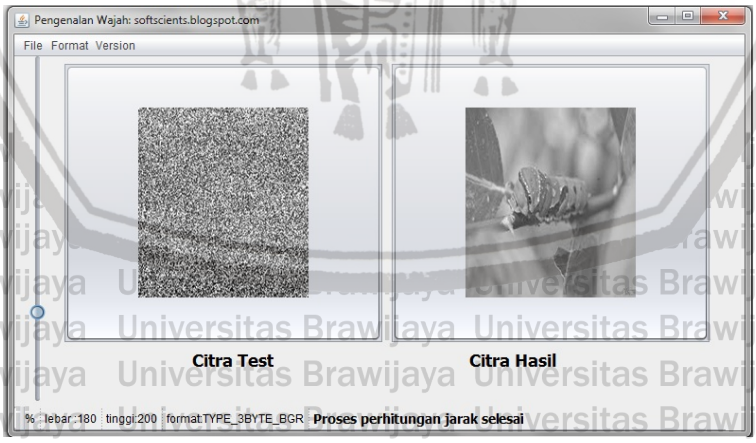
Tabel 6.6 Hasil pengujian kualitas citra digital

No	Presentase Noise	Nama Citra	Hasil yang Diharapkan	Hasil pengujian
1	0%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
2	10%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
3	20%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi

4	30%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
5	40%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
6	50%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
7	60%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
8	70%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
9	80%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
10	90%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
11	100%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
12	110%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
13	120%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
14	130%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
15	140%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
16	150%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
17	160%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
18	170%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
19	180%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
20	190%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
21	200%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
22	210%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
23	220%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
24	230%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
25	240%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
26	250%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
27	260%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
28	270%	Ulat Jengkal	Ulat Jengkal	Dapat Diprediksi

		Jagung	Jagung	
29	280%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
30	290%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
31	300%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
32	310%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
33	320%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
34	330%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
35	340%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
36	350%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
37	360%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
38	370%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
39	380%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
40	390%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi
41	400%	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Dapat Diprediksi

Hasil pengujian kualitas citra yang diberikan gradasi *noise* sebanyak 400% dapat dilihat pada Gambar 6.7.



Gambar 6.7 Hasil output jika *noise* citra 400%
Sumber : Pengujian

6.2.2 Analisis Hasil Pengujian Kualitas Citra

Proses analisa terhadap kualitas citra digital yang digunakan dilakukan dengan melihat performa aplikasi dalam melakukan prediksi pada citra digital yang telah diberi *noise* dengan

prosentase tertentu. Berdasarkan hal tersebut dapat diambil kesimpulan sebagai berikut : Pada pengujian kualitas citra digital yang diberi *noise* skala 0 – 400% maka sistem dapat melakukan prediksi citra dan hasil yang didapat sesuai data citra pada database.

6.2.3 Pengujian Validasi Citra Digital

6.2.3.1 Pengujian Validasi Citra Digital Dengan *Cropping*

Pada sistem pengolahan citra digital ini menggunakan *input* citra hama tanaman jagung yang kemudian dilakukan proses *cropping* untuk mengetahui apakah sistem tetap dapat memprediksi citra tersebut atau tidak. Pada gambar 6.8 akan diperlihatkan citra test yang sudah dipotong dan belum.

Gambar 6.8 Hasil citra test di-*crop*

Sumber : Pengujian

Hasil pengujian kualitas citra yang diberikan proses *cropping* pada citra *test* dapat dilihat pada Gambar 6.9.

Gambar 6.9 Hasil output jika citra test di-*crop*

Sumber : Pengujian

6.2.3.2 Analisis Hasil Pengujian Validasi Citra Digital Dengan *Cropping*

Proses analisa terhadap validasi citra digital yang digunakan dilakukan dengan melihat performa aplikasi dalam melakukan prediksi pada citra test yang telah di-*crop* sesuai dengan bentuk hama. Berdasarkan hal tersebut dapat diambil kesimpulan sebagai berikut : Pada pengujian validasi citra test yang di-*crop* maka sistem tidak dapat melakukan prediksi citra

karena ada beberapa bagian hama belalang yang hampir menyerupai bagian tubuh hama ulat jengkal jagung.

6.2.3.3 Pengujian Validasi Citra Digital Dengan Menyelipkan Objek Pada Citra Test

Pada sistem pengolahan citra digital ini menggunakan *input* citra hama tanaman jagung yang kemudian dilakukan proses menambah image tambahan pada citra test untuk mengetahui apakah sistem tetap dapat memprediksi citra tersebut atau tidak. Pada gambar 6.10 akan diperlihatkan citra test asli dan citra yang diberikan image tambahan.

Gambar 6.10 Citra yang ditambah objek kutu

Sumber : Pengujian

Tabel 6.7 Hasil pengujian Validasi Citra Digital Dengan Menyelipkan Objek Pada Citra Test

No.	Nama Citra	Hasil Pengujian	Hasil Prediksi
1.	Belalang	Ulat Jengkal Jagung	Tidak Dapat Dipediksi
2.	Kumbang Bubuk	Kumbang Bubuk	Dapat Dipediksi
3.	Ulat Grayak	Ulat Jengkal Jagung	Tidak Dapat Dipediksi
4.	Penggerek Batang Jagung	Ulat Jengkal Jagung	Tidak Dapat Dipediksi
5.	Ulat Jengkal Jagung	Ulat Jengkal Jagung	Tidak Dapat Dipediksi

Hasil pengujian kualitas citra pada data citra *test* diberikan 1 objek tambahan dapat dilihat pada Gambar 6.11.

Gambar 6.11 Hasil *Output* Citra Test yang ditambah objek kutu

Sumber : Pengujian

6.2.3.4 Analisis Hasil Pengujian Validasi Citra Digital Dengan Menyelipkan Satu Objek Pada Citra Test

Proses analisa terhadap validasi citra digital yang digunakan dilakukan proses menambah satu objek lain pada citra test untuk mengetahui apakah sistem tetap dapat memprediksi citra tersebut atau tidak. Berdasarkan hal tersebut dapat diambil kesimpulan sebagai berikut : Pada pengujian validasi citra *test* yang ditambah satu objek lain maka sistem dapat melakukan prediksi citra terbatas pada beberapa hama, yaitu kutu sedangkan hama belalang, ulat grayak, ulat jengkal jagung, penggerek batang jagung yang memiliki bagian tubuh hampir sama tidak dapat diprediksi sesuai dengan data training.

BAB VII
PENUTUP

7.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut:

1. Aplikasi sistem pengolahan citra digital untuk mendeteksi hama dan penyakit jagung ini menggunakan metode Principal Component Analysis (PCA). Implementasi PCA pada sistem ini dilakukan untuk menguji berbagai pose dari citra hama tanaman jagung tersebut.

2. Dalam pengujian validasi citra test yang diberikan satu objek tambahan menunjukkan bahwa sistem tetap dapat melakukan prediksi yang sesuai hanya pada 1 hama yaitu kutu, sedangkan pada 4 hama yaitu belalang, ulat grayak, ulat jengkal jagung, penggerek batang jagung tidak dapat diprediksi sesuai dengan data training karena ada bagian tubuh hama atau pose foto yang hampir mirip satu sama lain.

3. Semakin tinggi dari nilai *noise* yang terdapat pada citra hama, sistem tetap dapat memprediksi citra tersebut dan hasil prediksi tersebut sesuai dengan data training dikarenakan data yang diberikan noise tersebut telah terdeteksi bahwa gambar tersebut adalah ulat jengkal jagung walaupun bentuk ulat tersebut tidak terlihat.

7.2 Saran

Saran yang dapat diberikan untuk pengembangan sistem ini antara lain:

1. Untuk pengembangan lebih lanjut sistem ini dapat dikembangkan dengan menggunakan lebih banyak data pelatihan dan dijadikan sebuah perangkat lunak berbasis web dengan bahasa pemrograman *Java Server Pages* (JSP).
2. Untuk pengembangan lebih lanjut aplikasi ini dapat ditambah dengan metode lain agar gambar yang terdeteksi lebih bervariasi dan hasil yang didapat lebih akurat meskipun dengan spesifikasi perangkat keras minimum.
3. Untuk pengembangan lebih lanjut dapat dilakukan penambahan algoritma agar perangkat lunak dapat melakukan pengolahan citra lebih banyak lagi dari sisi yang

berbeda untuk data *training* dan data *test*, misalnya foto belalang dari arah depan, belakang dll.

DAFTAR PUSTAKA

[BAN-10] Bangun, Melly BR. 2010. *Analisis Kinerja Metode Canny Dalam Mendeteksi Tepi Karies Gigi*. Medan: Fakultas Matematika Dan Ilmu Pengetahuan Alam Universitas Sumatera Utara

[BOO-05] Booch, Grady, James Rumbaugh, Ivar Jacobson. 2005. *The Unified Modelling Language User Guide: Second Edition*. Addison Wesley.

[EDY-09] Edyana, Fajara. 2009. *Pemanfaatan Jaringan Syaraf Tiruan Untuk Pengenalan Rambu Lalu Lintas*. Jakarta: Fakultas Ilmu Komputer Universitas Pembangunan Nasional Veteran

[HAR-06] Hardiyanto, Budi. 2006. *Sistem Pakar Penentuan Kesesuaian Lahan Untuk Pemilihan Wilayah Budidaya Komoditas Pertanian (Studi Kasus: Kecamatan Klari, Karawang, Jawa Barat)*. Bogor: Fakultas Teknologi Pertanian Institut Pertanian Bogor.

[KUR-08] Kurbel, Karl E. 2008. *The Making of Information Systems:Software Engineering and Management in a Globalizaed World*. Springer.

[PRE-01] Pressman, Roger S. 2001. *Software Engineering: a practitioner's approach,7th edition*. Mc Graw Hill.

[PUR-10] Purnomo, Mauridhi Hery, Arif Muntasa. 2010. *Konsep Pengolahan Citra Digital Dan Ekstraksi Fitur*. Graha Ilmu.

[TAU-09] Taurisna, Afnisyah. 2009. *Analisis Pengaruh Kualitas Resolusi Citra Terhadap Kinerja Metode Pendeteksi Tepi*. Medan: Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Sumatera Utara.

[WAT-11] Wati, Riza Masitha. 2011. *Pengenalan Pola Senyum Menggunakan Backpropagatation Berbasis Ekstraksi Fitur Principal Component Analysis.Bangkalan: Jurusan Teknik Informatika Fakultas Teknik Universitas Trunojoyo.*