BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Analisis sistem merupakan tahap yang bertujuan untuk memahami sistem, mengetahui kekurangan sistem, dan menentukan kebutuhan perbandingan proses dan hasil pada sistem pengenalan pola wajah (*face recognition*). Dengan menganalisis prosedur sistem yang digunakan dan melakukan pengujian hasil, maka sistem dapat dievaluasi sehingga dapat dijadikan sebagai acuan untuk dalam proses pembentukan kesimpulan dan rekomendasi.

Pada tahap ini akan dibahas perbandingan proses dan hasil pada sistem pengenalan pola wajah menggunakan dua metode yaitu, metode *Eigenface*, yang menggunakan algoritma PCA (*Principal Component Analysis*), dan metode *Fisherface*, yang menggunakan algoritma LDA (*Linear Discriminant Analysis*)

Sedangkan yang menjadi tujuan dari pembuatan aplikasi perangkat lunak komputer (*software*) adalah untuk mengimplementasikan metode *Eigenface* dan metode *Fisherface* pada sistem pengenalan pola wajah. Kemudian, menguji performa atau kemampuan metode *Eigenface* dan metode *Fisherface* dalam melakukan pengenalan pola wajah pada keadaan yang telah dikondisikan. Dan pada akhirnya *software* atau perangkat lunak ini akan menghasilkan sebuah hasil pengenalan pola wajah yang dapat dianalisis kemudian untuk dibuat kesimpulan dan rekomendasi, sebagai hasil dari penelitian, sehingga dapat diketahui metode

mana yang efektif dalam sistem pengenalan pola wajah yang dapat digunakan dalam keadaan yang telah dikondisikan sebelumnya.

Berdasarkan kebutuhan sistem dan hasil analisis sistem di atas, maka sistem yang akan dibuat harus dapat memenuhi fungsi-fungsi sebagai berikut:

1. Mengimplementasikan metode *Eigenface* dan metode *Fisherface* pada sistem pengenalan pola wajah.
2. Menguji performa dan kemampuan metode *Eigenface* dan metode *Fisherface* pada sistem pengenalan pola wajah dalam keadaan yang telah dikondisikan sebelumnya.
3. Menghasilkan sebuah kesimpulan dan rekomendasi metode mana yang efektif dalam sistem pengenalan pola wajah yang dapat digunakan dalam keadaan yang telah dikondisikan sebelumnya.

3.2 Analisis Masalah

Berdasarkan hasil analisis terhadap sistem yang ada, maka diidentifikasikan masalah dari sistem yang ada adalah sebagai berikut :

1. Harus dapat mengemplementasikan metode *Eigenface* dan metode *Fisherface*.
2. Harus mampu menguji performa dan kedua algoritma yang digunakan pada kondisi yang telah ditentukan.
3. Harus bisa menghasilkan kesimpulan dan rekomendasi mengenai metode mana yang efektif dalam sistem pengenalan pola wajah dalam keadaan yang telah dikondisikan sebelumnya.

3.3 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional adalah sebuah langkah dimana seorang pembangun perangkat lunak (*software developer*) menganalisis sumber daya manusia yang akan menggunakan perangkat lunak yang dibangun, perangkat keras dan perangkat lunak yang dimiliki sesuai dengan kebutuhan atau permintaan, sehingga dapat ditentukan kompabilitas aplikasi yang dibangun terhadap sumber daya yang ada.

Setelah melakukan analisis non fungsional, maka dilanjutkan ke langkah berikutnya yaitu menentukan kebutuhan non fungsional sistem yang akan dibangun untuk disesuaikan dengan fakta yang ada.

Apabila terjadi ketidakcocokan antara fakta dan kebutuhan maka perlu adanya penyesuaian fakta terhadap kebutuhan yang ada. Apabila kebutuhan tidak dipenuhi maka sistem yang dibangun tidak akan berjalan baik sesuai yang diharapkan.

Analisis kebutuhan non fungsional yang dilakukan dibagi dalam tiga tahap, yaitu:

1. Analisis perangkat keras (*hardware*).
2. Analisis perangkat lunak (*software*).
3. Analisis pengguna (*user*).

3.3.1 Analisis Perangkat Keras

Spesifikasi minimum perangkat keras yang dapat digunakan adalah sebagai berikut:

1. PC atau Laptop dengan spesifikasi :
   1) Processor Intel Pentium 166 Mhz atau yang setara
   2) RAM 256 MB
   3) VGA dengan VRAM 8 MB
   4) Harddisk dengan ruang kosong sebesar 500 MB
2. *Webcam* dengan parameter :
   1) Resolution up to 5 *megapixel*
   2) Video Mode 8 *bit or higher*
   3) Interface USB Port
   4) Transmisison Rate : 640 X 480 = 30 *frame/second*

3.3.2 Analisis Perangkat Lunak

Perangkat lunak yang akan digunakan untuk tahap implementasi :

1) Sistem Operasi *Windows* 2000 / XP / Vista / 7
2) *Mathworks Matlab 7.1* sebagai aplikasi pembangun.
3) *Microsoft Visua*l C++ sebagai *compiler.*
4) *Star* UML sebagai tools pemodelan objek.

3.4 Analisis Algoritma

Analisis algoritma digunakan untuk mengetahui cara kerja masing-masing algoritma terhadap pengenalan citra *image* masukan.

3.4.1 *Preprocessing*

Citra digital yang dimuat memiliki intensitas *pixel* yang beragam. Tahap *preprocessing* berguna untuk membatasi intensitas *pixel* citra.

3.4.1.1 Konversi *image* ke *grayscale*

Sebelum citra diproses dengan masing-masing algoritma, citra wajah diproses dengan proses *training*. Proses *training* dilakukan dengan mengubah *image* RGB menjadi *image grayscale* untuk mendapatkan sebuah vektor ciri dan mendapatkan jarak *distance* yang nantinya akan digunakan pada proses pengenalan dengan masing-masing algoritma.

Tahap *grayscaling* citra adalah kegiatan untuk menyederhanakan model citra. Tiga *layer* pada citra berwarna, R-*layer*, G-*layer*, dan B-*layer* diubah menjadi satu *layer* grayscale. Untuk mengubah citra berwarna yang memiliki nilai matrik masing-masing R, G, dan B menjadi citra grayscale dengan membagi jumlah ketiga layer, dengan persamaan dibawah ini :

$$f_0(x,y) = \frac{f_i^R(x,y) + f_i^G(x,y) + f_i^B(x,y)}{3} \qquad (3.1)$$

Gambaran proses konversi RBG ke *grayscale* melalui perhitungan manual. Misalkan kita memiliki citra berukuran 3 x 4 piksel dengan nilai-nilai RBGnya sebagai berikut :

Tabel 3.1 Konversi Manual RGB ke *Grayscale*

(x,y)	0	1	2	3
0	31 33 22	42 44 33	46 48 37	41 43 32
1	42 44 33	44 46 35	64 66 55	82 84 73
2	188 189 181	186 187 179	193 194 186	157 158 150

Untuk konversi warna menjadi RGB digunakan rumus :

$$f_0(0,0) = \frac{31+33+22}{3} = 28{,}7$$
$$\vdots$$
$$\vdots$$
$$f_0(3,4) = \frac{157+158+150}{3} = 155$$

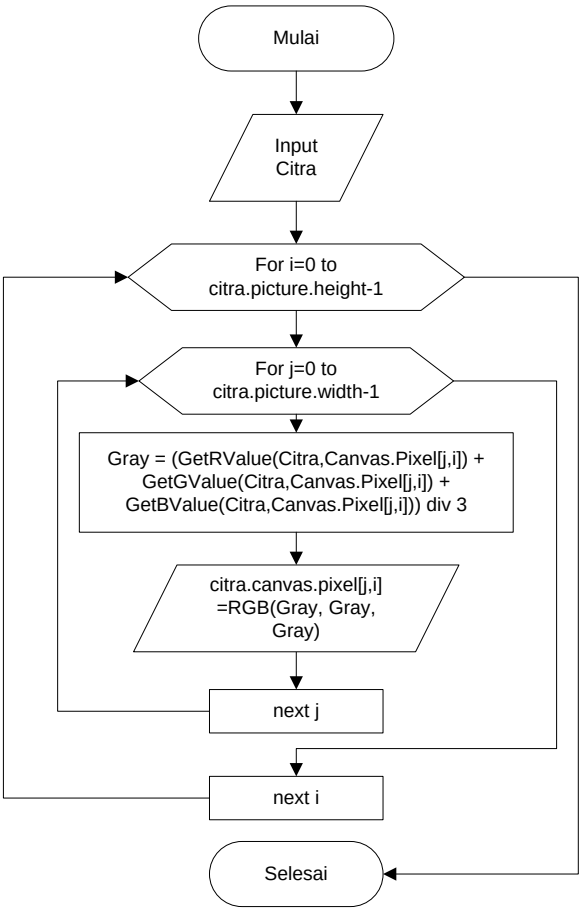
Perhitungan tersebut dilakukan untuk setiap koordinat, sehingga matriks citra hasil konversi seperti berikut :

Tabel 3.2 Konversi *Grayscale* Hasil Perhitungan Manual

(x,y)	0	1	2	3
0	29	40	44	39
1	40	42	62	80
2	186	184	191	155

Setelah didapatkan citra yang sudah ter-*grayscale*, selanjutnya citra diproses dengan mengubahnya menjadi vektor matriks untuk mendapatkan nilai vektor dari citra tersebut. Proses *grayscale* dapat dilihat pada *flowchart* di bawah ini :

fo adalah nilai *layer* baru yang didapatkan dari hasil perhitungan ketiga *layer* citra. Gambar 3.1 adalah *flowchart* algoritma yang digunakan untuk *grayscaling* citra.



Gambar 3.1 *Flowchart* proses *grayscale*

Keterangan:

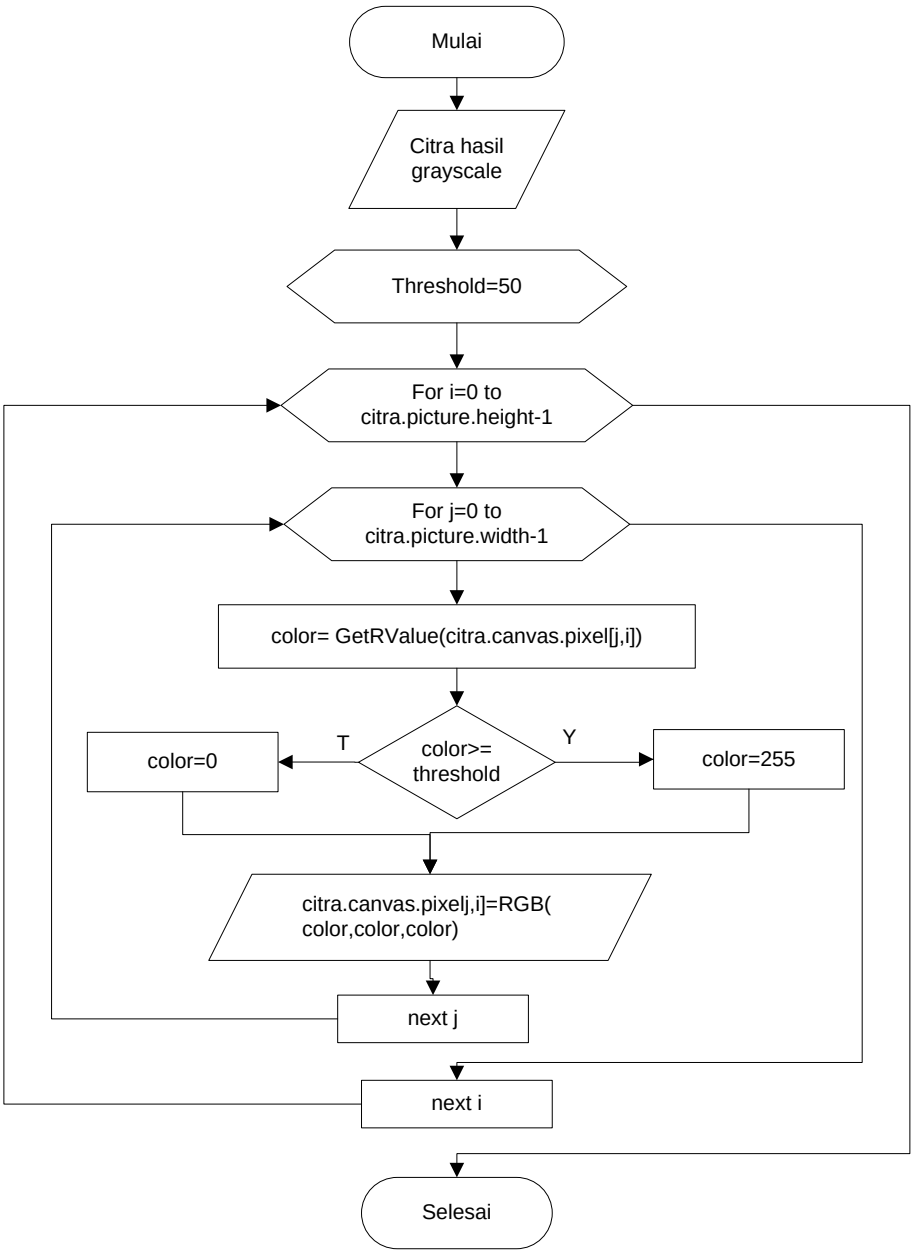
1. citra.picture.height : tinggi citra
2. citra.picture.width : lebar citra
3. citra.canvas.pixel[j,i] : fungsi yang digunakan untuk mengakses intensitas *pixel* dengan koodinat j,i.
4. GetRValue : fungsi yang digunakan untuk mendapatkan nilai layer R (*red*) suatu *pixel*.

5. GetGValue : fungsi yang digunakan untuk mendapatkan nilai layer G (*green*) suatu *pixel.*
6. GetBValue : fungsi yang digunakan untuk mendapatkan nilai layer B (*blue*) suatu *pixel.*
7. RGB(Gray, Gray, Gray) : fungsi yang digunakan untuk menggabungkan ketiga layer menjadi satu layer.

Hasil dari proses *grayscale* adalah citra dengan intensitas *pixel* sebanyak 256 intensitas. Nilai intensitas paling rendah merepresentasikan warna hitam dan intensitas paling tinggi merepresentasikan warna putih.

3.4.1.2 Proses *Tressholding Image*

Proses ini dilakukan untuk mengubah citra berderajat keabuan menjadi citra biner atau hitam putih. Intensitas *pixel* citra hasil *grayscale* dibandingkan dengan nilai ambang (*thresshold*). Jika nilai *pixel* lebih besar dari nilai ambang, maka *pixel*akan direpresentasikan dengan warna putih. Jika nilai *pixel* lebih kecil daripada nilai ambang, maka *pixel*akan direpresentasikan dengan warna hitam. Gambar 3.2 adalah *flowchart* algoritma yang digunakan pada proses binerisasi.



Gambar 3.2 *Flowchart* proses binerisasi.

Keterangan:

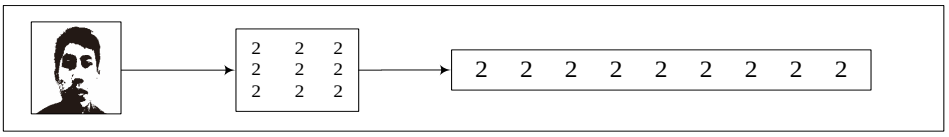
1. *Threshold* : nilai yang menjadi batas intensitas *pixel*. Jika suatu intensitas *pixel* kurang dari nilai ini, maka akan berwarna hitam. Jika lebih dari nilai *threshlod*, maka akan berwarna putih.
2. citra.picture.height : tinggi citra

3. citra.picture.width : lebar citra
4. citra.canvas.pixel[j,i]=clBlack?:apakah *pixel* yang sedang diakses berwarna hitam ?
5. GetRValue : fungsi yang digunakan untuk mendapatkan nilai layer R (*red*) suatu *pixel*.
6. RGB(Color,Color,Color) : fungsi yang digunakan untuk menggabungkan ketiga *layer* (Color,Color,Color) menjadi satu *layer*.

Berdasarkan *flowchart* pada gambar 3.2, jika nilai *pixel* lebih besar dari 50 maka *pixel*akan direprentasikan dengan warna putih dan sebaliknya. Nilai 50 didapatkan dari hasil percobaan dan merupakan nilai terbaik untuk binerisasi citra yang berpengaruh terhadap akurasi pada proses pengenalan.

3.4.1.3 *Reshape image* 2D ke vektor 1D

Setelah didapatkan *image* yang hasil *tresshold* selanjutnya ubah menjadi vektor satu dimensi untuk untuk mempermudah dalam menghitung *distance* dari setiap *image* yang nantinya digunakan dalam proses pengenalan. Misalnya dalam *training image* terdapat *image* dengan ukuran 3 x 4 piksel maka kita akan mempunyai *flatvector* ukuran 1 x 9. Ilustrasinya sebagai berikut :

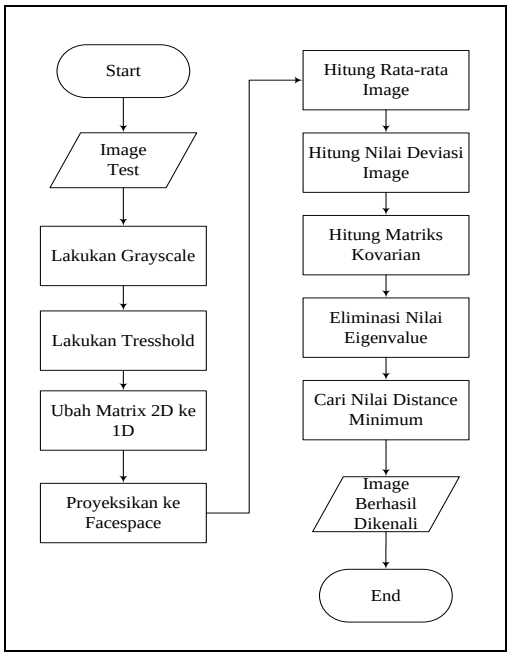


Gambar 3.3 Proses *Reshap*e 2D ke 1D vektor *image*

3.4.2 Algoritma *Eigenface*

Eigenface adalah salah satu algoritma pengenalan wajah yang didasarkan pada *Participal Component Analysis* (PCA) yang dikembangkan di MIT. Algoritma *eigenface* secara keseluruhan cukup sederhana. *Training image* direpresentasikan ke dalam sebuah *vectorflat* (gabungan vektor) dan digabung secara bersama-sama menjadi sebuah matriks tunggal. *Eigenfaces* dari masing-masing citra kemudian di ekstraksi dan disimpan dalam file *temporary* atau *database*. Test *image* yang masuk diidentifikasikan juga nilai *eigenface*-nya dan dibandingkan dengan *eigenface* dalam *database* atau file *temporary*.

Langkah-langkah dari proses proses pengenalan wajah *(recognition)* dari sebuah citra dengan metode *eigenface* terlihat pada *flowchart* (Gambar 3.4) dibawah ini :



Gambar 3.4 *Flowchart* Tahapan Proses Pengenalan Wajah Metode *Eigenface*

Adapun algoritma selengkapnya adalah:

1. Buat *MakeFlatVectors* (*ImageList, N, M*): Image List adalah kumpulan dari *N training image*, dimana setiap image adalah *W* x *H* piksel. *M* adalah ukuran dari *vector flat* yang harus dibuat.
2. Gabungkan setiap *image* dalam *WH* elemen *vector* dengan menggabungkan semua baris. Buat *image* matriks sebagai matriks *N* x *WH* berisi semua gambar yang digabung.
3. Jumlahkan semua baris pada *image* matriks dan bagi dengan N untuk mendapatkan rataan vektor gabungan. Namakan vektor elemen *WH* dengan *R*.
4. Kurangi *image* matriks dengan *average image R*. Namakan matriks baru ukuran *N* x *WH* sebagai *R*'.
5. Jika pada elemen-elemen dari matriks *R'* ditemukan nilai negative, ganti nilainya dengan nilai 0.

Kemudian identifikasi dilakukan dengan proyeksi dengan algoritma sebagai berikut:

1. Buat ProjectToFaceSpace(test_image): Image berukuran *W* x *H* Piksel.
2. Gabungkan elemen vektor *WH* dan namakan *img*.
3. Load nilai *R* dari database.
4. Kurangi img dengan *R* hingga didapat *img'*.
5. Jika pada *img'* ditemukan elemen dengan nilai negatif, ganti dengan nilai 0 untuk mendapatkan vektor ukuran *img"*.

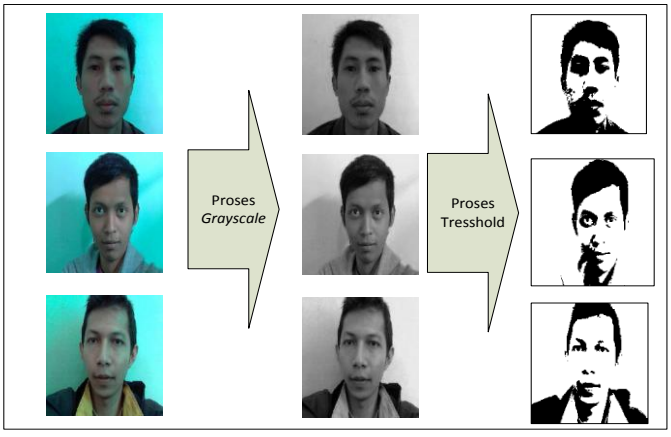
Proses terakir adalah identifikasi yaitu dengan memproyeksikan *test image* ke *face space* dan menghitung *distance*

1. Load semua wajah yang sudah diproyeksikan ke database
2. Proj=projectToFaceSpace(test_image).
3. Lakukan operasi pengurangan, proj dengan semua wajah yang telah diproyeksikan. Ambil nilai absolutnya dan jumlahkan, hasil adalah *distance*.
4. Ambil *distance* terkecil sebagai hasil dari wajah yang telah diproyeksikan. Wajah ini menjadi hasil identifikasi.

Untuk lebih jelasnya algoritma *eigenface* dapat diuraikan dalam ilustrasi sebagai berikut :

1. Proses *Image Test*

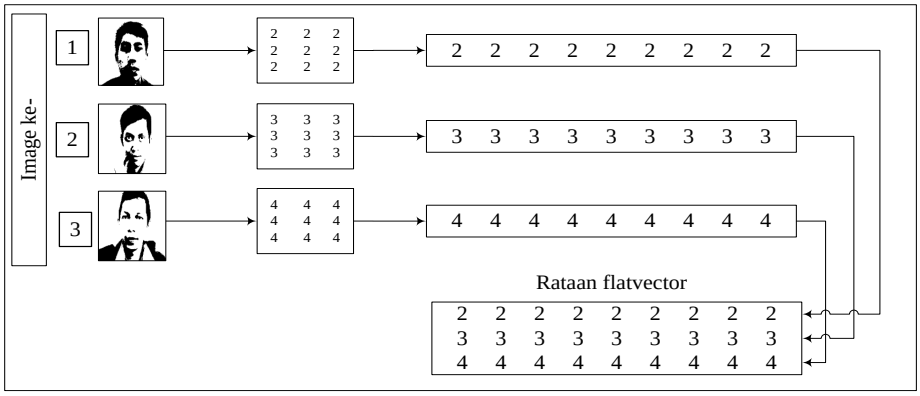
   Langkah pertama setelah didapatkan citra wajah hasil *capture*, citra wajah dirubah ke dari bentuk RGB ke dalam bentuk *grayscale*, setelah didapatkan citra keabuan ubah menjadi citra hitam putih dengan melakukan *tresshold* agar kompleksitas citra lebih sederhana.



Gambar 3.5 Proses Mengubah Citra RGB ke *grayscale* dan *tresshold*

2. Penyusunan *flatvector*

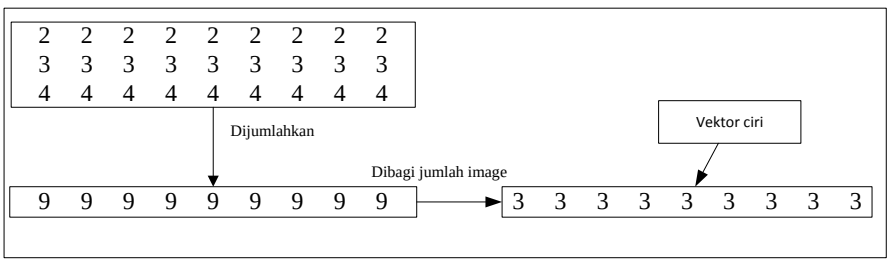
Selanjutnya adalah menyusun seluruh *training image* menjadi suatu matriks tunggal. Misalnya *image* yang disimpan berukuran *H* x *W* piksel dan jumlahnya *N* buah, maka akan dimiliki *vector* ciri dengan dimensi *N* x (*W* x *H*). Misalnya dalam *training image* terdapat 3 *image* dengan ukuran 3 x 4 piksel maka kita akan mempunyai *eigenvector* ukuran 3 x 9. Ilustrasinya sebagai berikut:



Gambar 3.6 Penyusunan flatvector

3. Perhitungan rataan *flatvector*

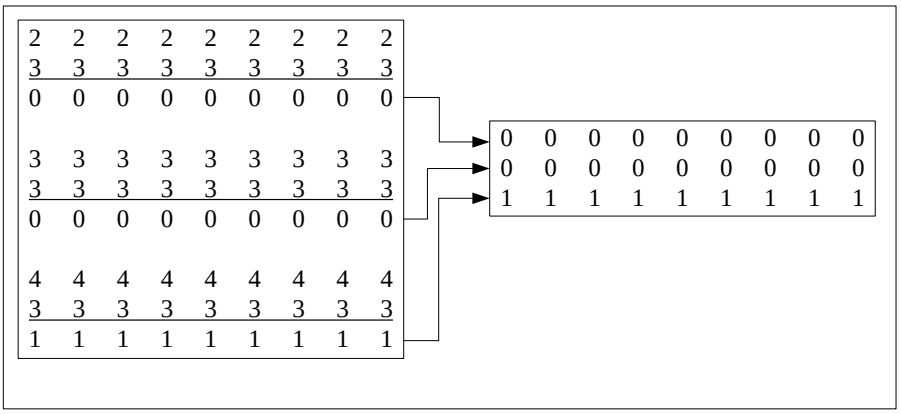
Dari *vector* ciri yang telah diperoleh, jumlahkan seluruh barisnya sehingga diperoleh matriks berukuran 1 x (*W* x *H*). Setelah itu bagi matriks tersebut dengan jumlah *image* *N* untuk mendapatkan nilai rata-rata *vector* ciri:



Gambar 3.7 Nilai rata-rata *flatfector*

4. Proyeksikan image ke *facespace*

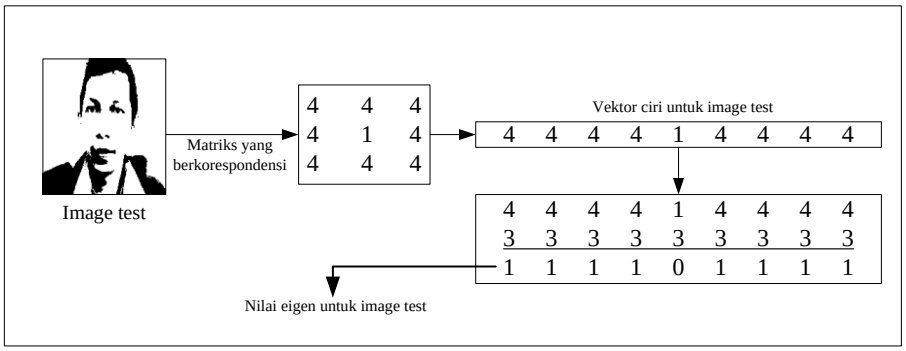
   Image diproyeksikan ke *facespace* dengan mengkalikan di basis *eigenface*. Proyeksi vektor wajah akan dibandingkan dengan vektor yang sesuai. Dengan menggunakan nilai rata-rata *vector* ciri, akan dihitung *eigenface* untuk matriks *vector* ciri yang telah disusun. Caranya dengan mengurangi baris-baris pada matriks *vector* ciri dengan nilai rata-rata *vector* ciri. Jika didapatkan nilai di bawah nol, ganti nilainya dengan nol.



Gambar 3.8 Perhitungan *eigenface*

5. Ektraksi PCA *image test*

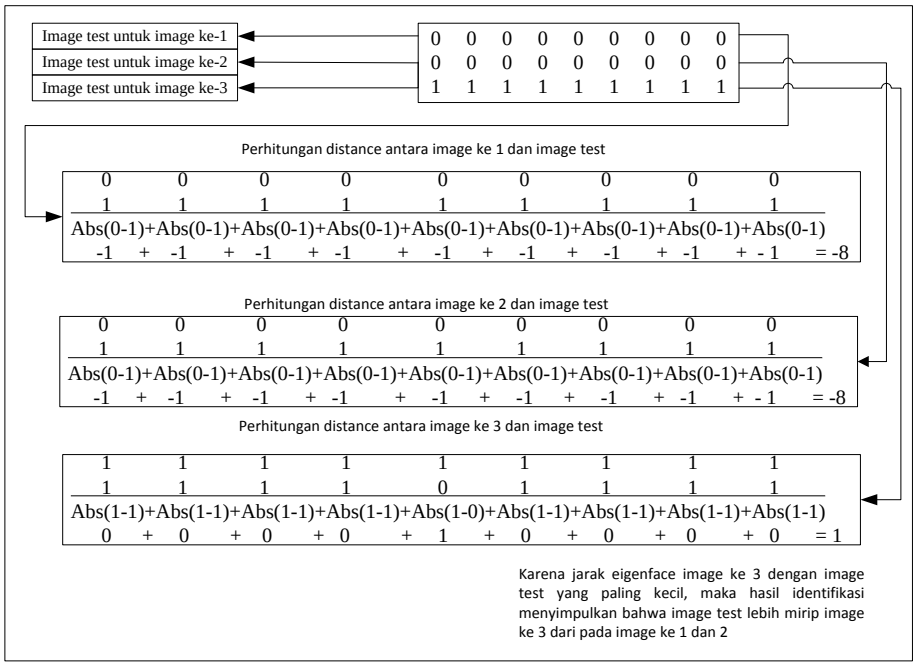
   Hasil proyeksi tersebut diektraksi dengan perhitungan PCA untuk mendapatkan *feature* dari image. *Feature* adalah komponen–komponen penting dari *image-image training* yang didapatkan dari proses *training*. *Feature* inilah yang nanti akan digunakan untuk mengidentifikasikan *image* yang akan dikenali. Kalkulasi nilai *eigenface* untuk matriks *testface*, dengan cara yang sama dengan penentuan *eigenface* untuk *vector* ciri.



Gambar 3.9 Perhitungan *eigenface* untuk *image test*

6. Cari *distance* minimum antara *image test* dan *image* hasil *training.*

   Bandingkan nilai *euclidean distance minimum* dari *image* yang di-*capture* dengan image yang sudah ada di *database*. Setelah nilai *eigenface* untuk *image test* diperoleh maka kita bisa melakukan identifikasi dengan menentukan jarak (*distance*) terpendek dengan *eigenface* dari *eigenvector training image*. Caranya dengan menentukan nilai *absolute* dari pengurangan baris *i* pada matriks *eigenface training image* dengan *eigenface* dari *testface*, kemudian jumlahkan elemen-elemen penyusun vektor yang dihasilkan dari pengurangan tadi dan ditemukan jarak *d* indeks *i*. Lakukan untuk semua baris. Cari nilai *d* yang paling kecil.



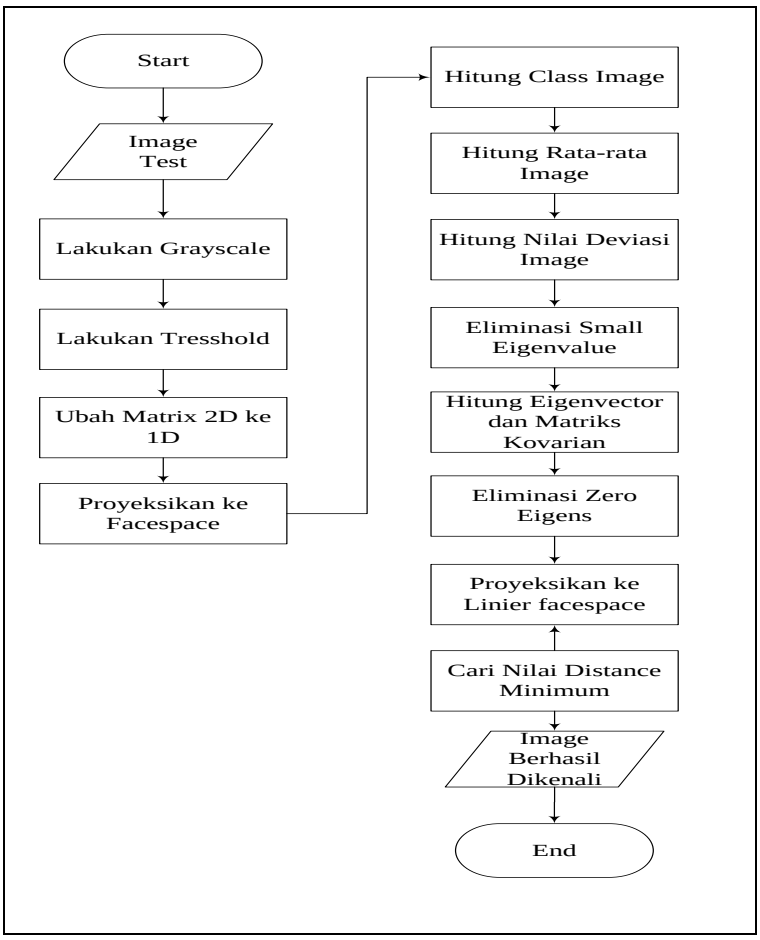
Gambar 3.10 Proses Indentifikasi dengan *Input Image test*

3.4.3 Algoritma *Fisherface*

Dasar dari metode *Fisherface* ini adalah *Fisher's Linear Discriminant* (FLD). FLD merupakan salah satu contoh metode *class spesific*, karena metode ini berusaha untuk membentuk jarak (*scatter*) antar kelas dan intra kelas sehingga dapat menghasilkan klasifikasi yang lebih baik.

Jika terdapat 20 sampel (N = 20) yang terbagi atas 2 kelas sampel. Dimensi sampel, n = 2 dan dimensi reduksi, m = 1. Jadi, sampel akan ditransformasikan dari ruang sampel dimensi-2 ke ruang ciri dimensi-1. Transformasi dilakukan dengan metode PCA dan FLD. Hasil proyeksi vektor ciri sampel pada ruang ciri dimensi-1 dengan metode PCA dan FLD.

Langkah-langkah dari proses pengenalan wajah *(recognition)* dari sebuah citra dengan metode *fisherface* terlihat pada *flowchart* (Gambar 3.11) dibawah ini :



Gambar 3.11 *Flowchart* Tahapan Proses Pengenalan Wajah Metode *Fisherface*

Konstruksi *fisherface* adalah pembuatan suatu set *fisherface* dari suatu set gambar *training* dengan menggunakan perhitungan PCA dan FLD. Perhitungan PCA dilakukan dengan langkah sebagai berikut :

1. Buat *MakeFlatVectors* (*ImageList, N, M*): Image List adalah kumpulan dari *N training image*, dimana setiap image adalah *W* x *H* piksel. *M* adalah ukuran dari *vector flat* yang harus dibuat.

2. Gabungkan setiap *image* dalam *WH* elemen *vector* dengan menggabungkan semua baris. Buat *image* matriks sebagai matriks *N* x *WH* berisi semua gambar yang digabung.
3. Jumlahkan semua baris pada *image* matriks dan bagi dengan N untuk mendapatkan rataan vektor gabungan. Namakan vektor elemen *WH* dengan *R.*
4. Kurangi *image* matriks dengan *average image R*. Namakan matriks baru ukuran *N* x *WH* sebagai *R*'.

Kemudian identifikasi dilakukan dengan menggunakan perhitungan FLD sebagai berikut:

1. Buat ProjectToFaceSpace(test_image): Image berukuran *W* x *H* Piksel.
2. Transformasikan *training set* ke dalam vektor kolom.
3. Bentuk *average face* dari *facespace*
4. Hitung nilai deviasi *image.*
5. Eliminasi nilai terkecil *eigenvalue*.
6. Hitung *eigenvector* dan matriks kovarian.
7. Hitung matriks proyeksi *fisher* dengan mengurutkan vektor eigen berdasarkan besarnya nilai *eigen* masing-masing vektor *eigen* dan mengambil komponen vektor *eigen* yang memiliki nilai *eigen* tidak nol.
8. Hitung bobot tiap *fisherface* terhadap masing-masing gambar wajah pada *training set* dengan memproyeksikan nilai deviasi *facespace* terhadap *average face* kedalam matriks proyeksi optimal.

Proses terakir adalah identifikasi yaitu dengan memproyeksikan *test image* ke *face space* dan menghitung *score*.

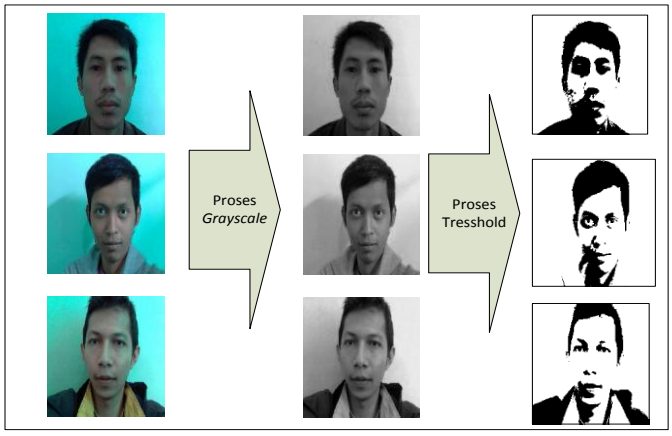
1. Load semua wajah yang sudah diproyeksikan ke database
2. Proj=projectToFaceSpace(test_image).
3. Lakukan operasi pengurangan, proj dengan semua wajah yang telah diproyeksikan. Ambil nilai absolutnya dan jumlahkan, hasil adalah *distance*.

4. Hitung jarak *distance* antara bobot *image test* dan bobot *training set.*
5. Ambil *distance* terkecil sebagai hasil dari wajah yang telah diproyeksikan. Wajah ini menjadi hasil identifikasi.

Untuk lebih jelasnya algoritma *fisherface* dapat diuraikan dalam ilustrasi sebagai berikut :

1. Proses *Image Test*

   Langkah pertama setelah didapatkan citra wajah hasil *capture*, citra wajah dirubah ke dari bentuk RGB ke dalam bentuk *grayscale*, setelah didapatkan citra keabuan ubah menjadi citra hitam putih dengan melakukan *tresshold* agar kompleksitas citra lebih sederhana.

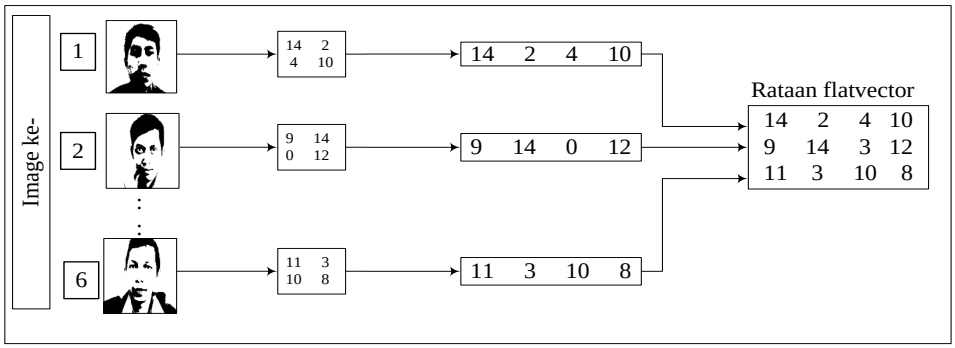


Gambar 3.12 Proses Mengubah Citra RGB ke *grayscale* dan *tresshold*

2. Penyusunan *flatvector*

   Langkah pertama adalah menyusun seluruh *training image* menjadi suatu matriks tunggal. Misalnya *image* yang disimpan berukuran *H* x *W* piksel dan jumlahnya *N* buah, maka akan dimiliki *vector* ciri dengan dimensi *N* x (*W* x *H*). Misalnya dalam *training image* terdapat 3 *image* dengan ukuran 3 x 4

piksel maka kita akan mempunyai *eigenvector* ukuran 3 x 9. Ilustrasinya sebagai berikut :



Gambar 3.13 Penyusunan *flatvector*

3. Perhitungan Tingkat Kemiripan

Dari *vector* ciri yang telah diperoleh, jumlahkan seluruh barisnya sehingga diperoleh matriks berukuran 1 x (*W* x *H*). Setelah itu bagi matriks tersebut dengan jumlah *image* *N* untuk mendapatkan nilai rata-rata *vector* ciri, ilustrasinya sebagai berikut :

Misalkan kita memiliki data sample random sejumlah 7 orang dengan 2 pose.

Tabel 3.3 Ciri dari 7 Orang, Masing-masing 2 pose

Orang ke	j	$f_{j,1}$	$f_{j,2}$	$f_{j,3}$	$f_{j,4}$
1	1	14	4	4	10
	2	3	2	2	5
2	3	9	0	0	12
	4	7	11	11	7
3	5	13	6	6	10
	6	11	13	13	6
4	7	6	6	6	4
	8	0	6	6	2
5	9	12	12	12	2
	10	6	7	7	10
6	11	9	3	3	4
	12	11	10	10	8
7	13	13	12	12	2
	14	11	0	0	10

Misalkan citra yang diuji mempunyai ciri ke 1, 2, 3 dan 4 dan masing-masing adalah 7, 6, 10, 9. Maka beda kuadrat dari masing-masing ciri pelatihan terhadap citra uji dapat dilihat pada tabel 3.4.

Dengan persamaan

$$(f_{j,i} - f_{uji,i})^2 \qquad (3.2)$$

Maka didapatkan nilai beda kuadrat dari masing-masing ciri pelatihan terhadap citra uji.

Tabel 3.4 Beda Kuadrat dari masing-masing ciri pelatihan Terhadap Citra Uji

No	Ciri ke-1	Ciri ke-2	Ciri ke-3	Ciri ke-4
1	(14-7)²	(2-6)²	(4-10)²	(10-9)²
2	(3-7)²	(6-6)²	(2-10)²	(5-9)²
3	(9-7)²	(14-6)²	(0-10)²	(12-9)²
4	(7-7)²	(13-6)²	(11-10)²	(7-9)²
5	(13-7)²	(6-6)²	(6-10)²	(10-9)²
6	(11-7)²	(13-6)²	(13-10)²	(6-9)²
7	(6-7)²	(0-6)²	(6-10)²	(4-9)²
8	(0-7)²	(5-6)²	(6-10)²	(2-9)²
9	(12-7)²	(12-6)²	(12-10)²	(2-9)²
10	(6-7)²	(0-6)²	(7-10)²	(10-9)²
11	(9-7)²	(2-6)²	(3-10)²	(4-9)²
12	(11-7)²	(3-6)²	(10-10)²	(8-9)²
13	(13-7)²	(2-6)²	(12-10)²	(2-9)²
14	(11-7)²	(9-6)²	(0-10)²	(10-9)²

Setelah didapatkan nilai kuadrat dari masing-masing ciri pelatihan terhadap citra uji. Jumlahkan secara keseluruhan pada tiap-tiap ciri pada masing-masing citra dengan persamaan.

$$(f_{j,1} - f_{uji,1})^2 + (f_{j,2} - f_{uji,2})^2 + (f_{j,3} - f_{uji,3})^2 + (f_{j,4} - f_{uji,4})^2 \qquad (3.3)$$

Sehingga diperoleh nilai seperti pada Tabel 3.5 dibawah ini:

Tabel 3.5 Hasil Penjumlahan Beda Kuadrat Dari Masing-Masing Ciri Pelatihan Terhadap Citra Uji

No	Ciri ke-1	Ciri ke-2	Ciri ke-3	Ciri ke-4	Jumlah
1	49	16	36	1	102
2	16	0	64	16	96
3	4	64	100	9	177
4	0	49	1	4	54
5	36	0	16	1	53
6	16	49	9	9	83
7	1	36	16	25	78
8	49	1	16	49	115
9	25	36	4	49	114
10	1	36	9	1	47
11	4	16	49	25	94
12	16	9	0	1	26
13	36	16	4	49	105
14	16	9	100	1	126

Hasil perhitungan pada Tabel 3.5 selanjutnya diakarkan, maka akan menghasilkan nilai seperti terlihat pada tabel 3.6. pada abel 3.6 tampak bahwa nilai minimal terletak pada data ke 12, data ke 12 merupakan orang ke 6 pose ke 2. Sehingga data yang diuji tersebut dikenali sebagai orang yang ke 6 pose ke 2

4. Cari *distance* minimum antara *image test* dan *image* hasil *training.*

   Bandingkan nilai *euclidean distance minimum* dari *image* yang di-*capture* dengan image yang sudah ada di *database*. Setelah nilai *eigenface* untuk *image test* diperoleh maka kita bisa melakukan identifikasi dengan menentukan jarak (*distance*) minimum dengan *eigenface* dari *eigenvector training image*.

   Dengan persamaan

$$d_v = \sum_{j-1}^{m} \| F_{trainke\ j,w} - F_{uji,w} \| \qquad (3.4)$$

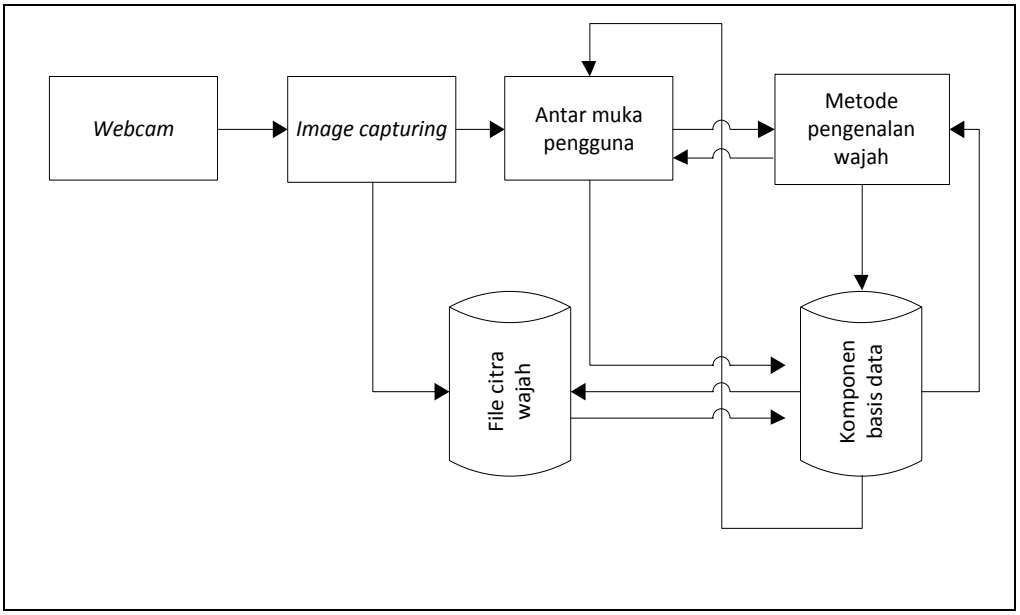
Maka akan didapatkan nilai distance minimum dari setiap pose *image*. Seperti terlihat pada Tabel 3.6 dibawah ini :

Tabel 3.6 Hasil Akhir Perhitungan Tingkat Kemiripan Menggunakan *Eucledean Distance*

No	Jumlah Ciri	Nilai *Eucledean Distance*	Hasil
1	102	10.1	
2	96	9.798	
3	177	13.304	
4	54	7.3485	
5	53	9.1104	
6	83	8.8318	
7	78	10.724	
8	115	6.8557	
9	114	9.9654	
10	47	**5.099**	**Minimal**
11	94	10.247	
12	26	11.225	

3.5 Analisis Kebutuhan Sistem

Aplikasi perbandingan algoritma metode *Fisherface* dan metode *Eigenface* dalam pengenalan pola wajah terdiri dari beberapa komponen utama yang dapat digambarkan secara umum dalam suatu model seperti tampak pada (Gambar 3.15) dibawah ini.



Gambar 3.14 Hubungan Antar Sub Sistem Pada Aplikasi Secara Menyeluruh

Keterangan :

Komponen *Webcam* : Piranti masukkan yang digunakan dalam aplikasi perbandingan algoritma ini adalah *webcam*. *Webcam* digunakan untuk mengambil citra wajah, dimana citra wajah ini akan disimpan ke dalam *databas,* yang nantinya akan dicocokan dengan citra wajah yang di-*capture* saat proses deteksi.

Komponen *Image Capturing* : Komponen ini berfungsi melakukan mekanisme pengambilan citra wajah dengan media *webcam*, baik untuk disimpan

sebagai file citra wajah maupun untuk citra wajah yang digunakan sebagai *input* pada saat proses deteksi wajah.

File Citra Wajah : citra wajah yang digunakan adalah citra wajah hasil *capture* dan hasil pengolahan pada proses *training.*

Komponen *Basis data* : komponen ini berfungsi untuk menyimpan data-data yang dibutuhkan oleh aplikasi perbandingan algoritma.

Sistem Pengenalan Wajah : Pengenalan wajah dilakukan dengan mencocokan citra wajah yang di-*capture* pada saat proses *training* ataupun pada saat proses deteksi wajah dengan menggunakan metode *fisherface* dengan metode *eigenface*.

3.6 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional merupakan tahap pendefinisian layanan / fungsi / proses yang harus disediakan oleh sistem, bagaimana reaksi sistem terhadap input dan apa yang harus dilakukan oleh sistem pada situasi tertentu.

Berdasarkan analisis terhadap permasalahan yang ada, maka fungsional-fungsional yang terdapat pada aplikasi ini adalah proses pengenalan pola wajah. Analisis yang dilakukan dimodelkan dengan menggunakan *Unified Modeling Language* (UML). Tahap-tahap pemodelan dalam analisis berbasis objek tersebut antara lain identifikasi aktor, *use case diagram, sequence diagram, activity diagram* dan skenario.

3.6.1 Identifikasi Aktor

Aktor yang terlibat dalam sistem yaitu pengguna (*user*). Pengguna disini merupakan orang yang mengunakan sistem. Para pengguna aplikasi sistem pengenalan pola wajah (*face recognition*) yang akan dibangun harus memiliki karakteristik sebagai berikut :

A. Karakterisik Psikologi Pengguna

User mempunyai sikap *spatial* karena pada umunya *user* perangkat ini merupakan *user* yang sudah tentang teknik-teknik pemrosesan dalam suatu perangkat lunak.

Tabel 3.7 *User phsychological characteristic*

Cognitive style	*Attitude*	*Motivation*
Spatian/intuitive	*Netral*	*Moderate*

B. Pengalaman dan Pengetahuan Pengguna

Adapun pengetahuan dasar dan pengalaman yang harus dimiliki oleh user untuk menjalankan aplikasi ini adalah :

Tabel 3.8 User Knowledge and Experiance

Tingkat Pendidikan	**Kemampuan Membaca**	**Kecepatan Membaca**
Usia 18 - 40 tahun	Bisa membaca	*Medium*
Computer Literacy	Tugas User	Pengalaman *System*
Moderate	*User* diharuskan memiliki pengetahuan tentang pengolahan atau pemrosesan citra	*User* dapat mengoperasikan komputer dan webcam.
Pengalaman Aplikasi	Bahasa yang digunakan	*System* Lainnya

C. Karekteristik Fisik Pengguna

Merupakan kriteria fisik yang harus dimiliki *user*, agar mampu menjalankan aplikasi ini. Kriteria fisik ini merupakan bagian dari spesifikasi *user* yang sebagian besar tidak dapat diubah.

Tabel 3.9 *User Physical Characteristic*

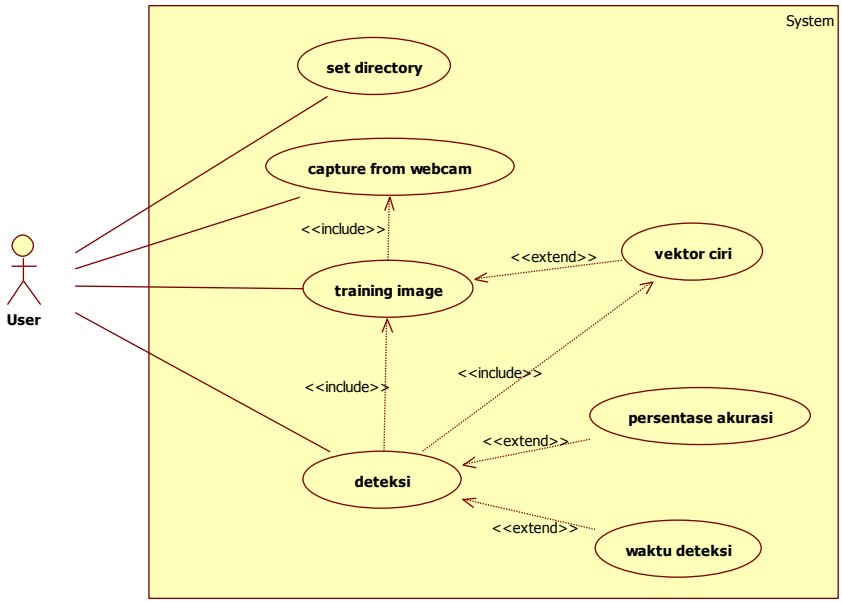
Buta Warna	Penggunaan Tangan	Jenis Kelamin
Tidak buta warna	*Ambidextrous*	Pria atau Wanita

3.6.2 Model *Use case*

Pemodelan aplikasi berorientasi objek digambarkan dengan diagram *use case*. Dalam diagram ini digambarkan proses yang terjadi pada aplikasi, yang berinteraksi langsung dengan *user*.

A. Diagram *Use case*

Pada Gambar 3.14, digambarkan diagram *use case* untuk perangkat lunak yang akan dibangun.



Gambar 3.15 Diagram *Use case*

B. Definisi Aktor

Aktor dalam hal ini adalah *user*. *User* merupakan orang dari berbagai kalangan dan umur yang mampu mengoperasikan komputer. *User* diharapkan mengetahui bahasa inggris tingkat dasar.

C. Definisi *Use case*

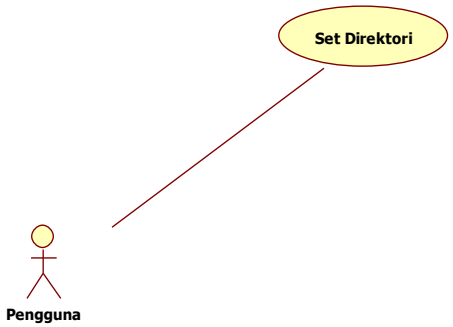
Definisi dari diagram *use case* pada Gambar 3.16, dideskripsikan secara singkat dalam Tabel 3.10 dibawah ini :

Tabel 3.10 Definisi *Use case*

No	*Use case*	Deskripsi
a.	*Set Directory*	Mengatur lokasi penyimpanan *image* hasil pengambilan citra dari *webcam.*
b.	*Capture Image*	Mengambil *sample* citra wajah dari *webcam.*
c.	*Training*	Mengolah citra wajah hasil *capture* dari *webcam* sebelum dilakukan pengenalan dan sebelum disimpan ke dalam *database*.
d.	Deteksi	Mengolah citra wajah hasil *capture* dari *webcam* untuk dilakukan pengenalan dengan citra wajah yang sudah tersimpan di dalam *database.*

3.6.3 *Use Case Diagram, Skenario, Activity Diagram, Sequence Diagram, Statechart Diagram Set Directory*

Skenario proses-proses yang terjadi pada aplikasi, dipaparkan dengan tujuan untuk lebih memudahkan dalam pembangunan aplikasi, agar lebih konsisten.

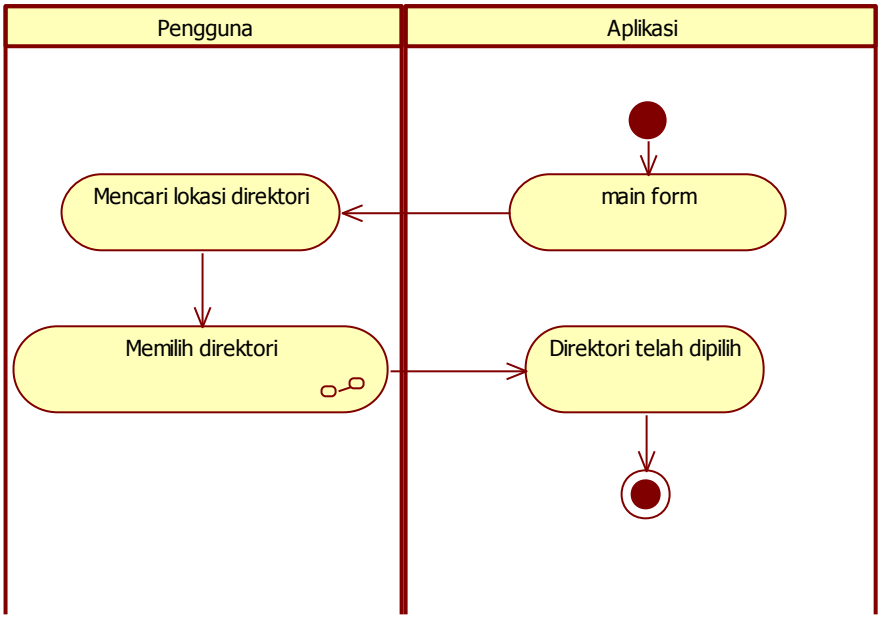


Gambar 3.16 ***Use case set directory***

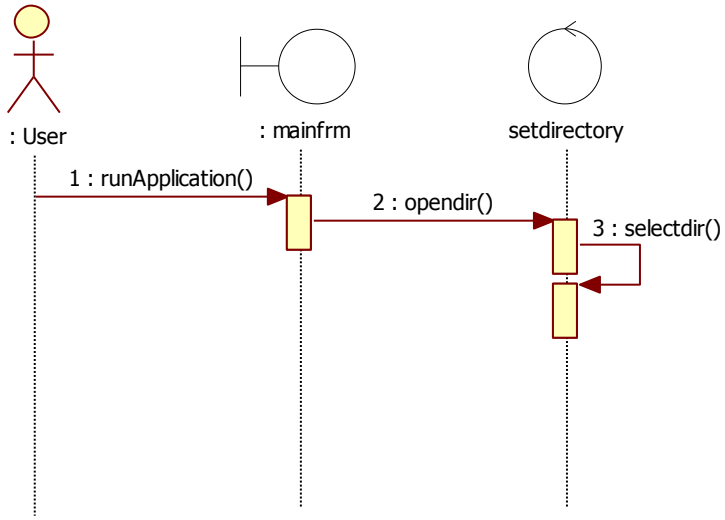
Fungsi ini melakukan pengaturan terhadap direktori yang akan digunakan untuk membandingkan *image* hasil *training* dan *image* uji. Skenario fungsi dapat dilihat pada Tabel 3.11.

Tabel 3.11 Skenario proses *set directory*

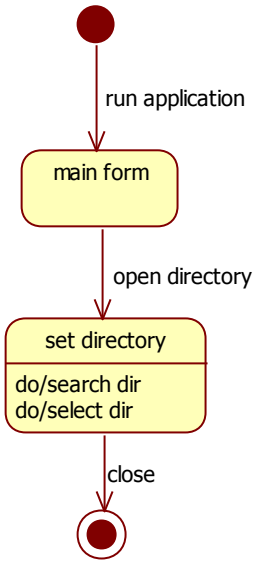
Identifikasi		
Nomor	1	
Nama	*Set Directory*	
Tujuan	Memastikan pengaturan lokasi untuk perbandingan *image*	
Deskripsi	Pengguna melakukan pengaturan terhadap direktori yang menyimpan *image* hasil *training*	
Aktor	Pengguna	
Skenario Utama		
Kondisi awal	Aplikasi baru dijalankan dan masih dalam kondisi awal	
Aksi Aktor		**Reaksi Sistem**
Pengguna melakukan pengaturan terhadap direktori yang menyimpan *image* hasil *training*		Direktori *image* telah diatur.
Kondisi akhir	Direktori *image* telah diatur.	



Gambar 3.17 ***Activity diagram set directory***



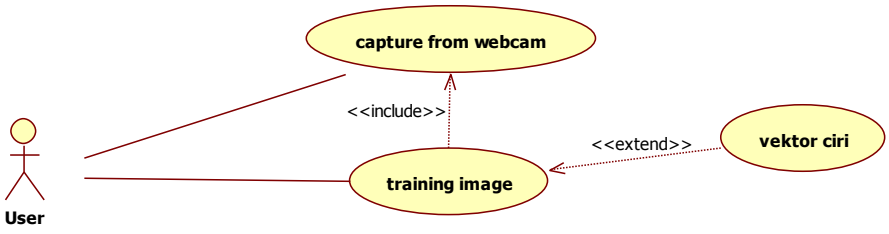
Gambar 3.18 ***Sequence Diagram Set Directory***



Gambar 3.19 ***Statechart set directory***

3.6.4 *Use Case Diagram, Skenario, Activity Diagram, Sequence Diagram, Statechart Diagram training image*

Skenario proses-proses yang terjadi pada aplikasi, dipaparkan dengan tujuan untuk lebih memudahkan dalam pembangunan aplikasi, agar lebih konsisten.

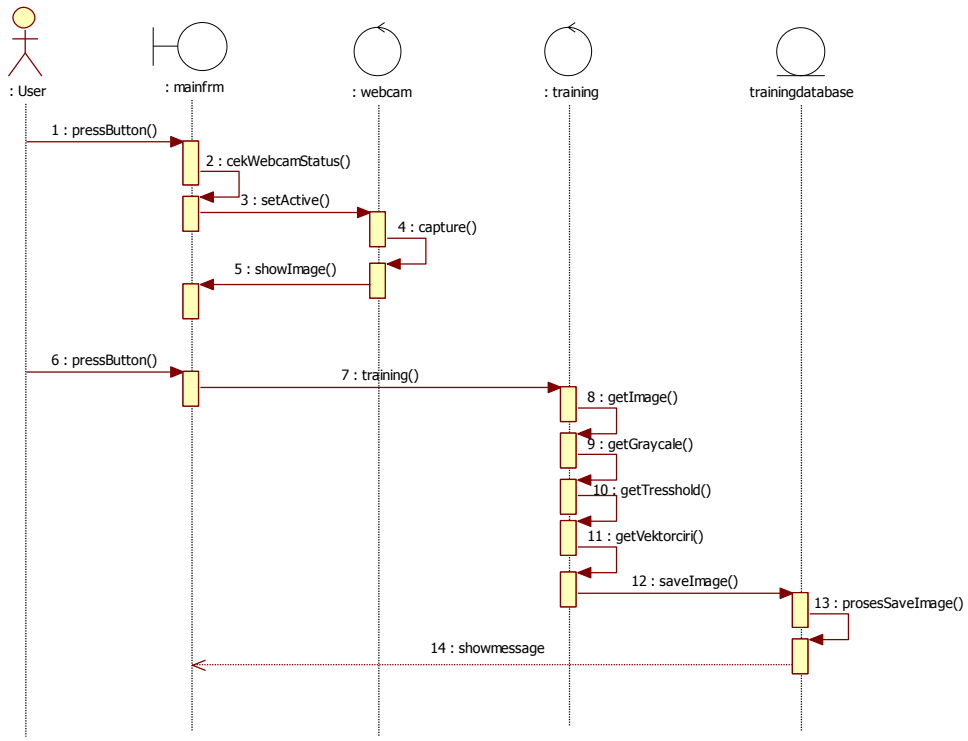


Gambar 3.20 ***Use case proses training image***

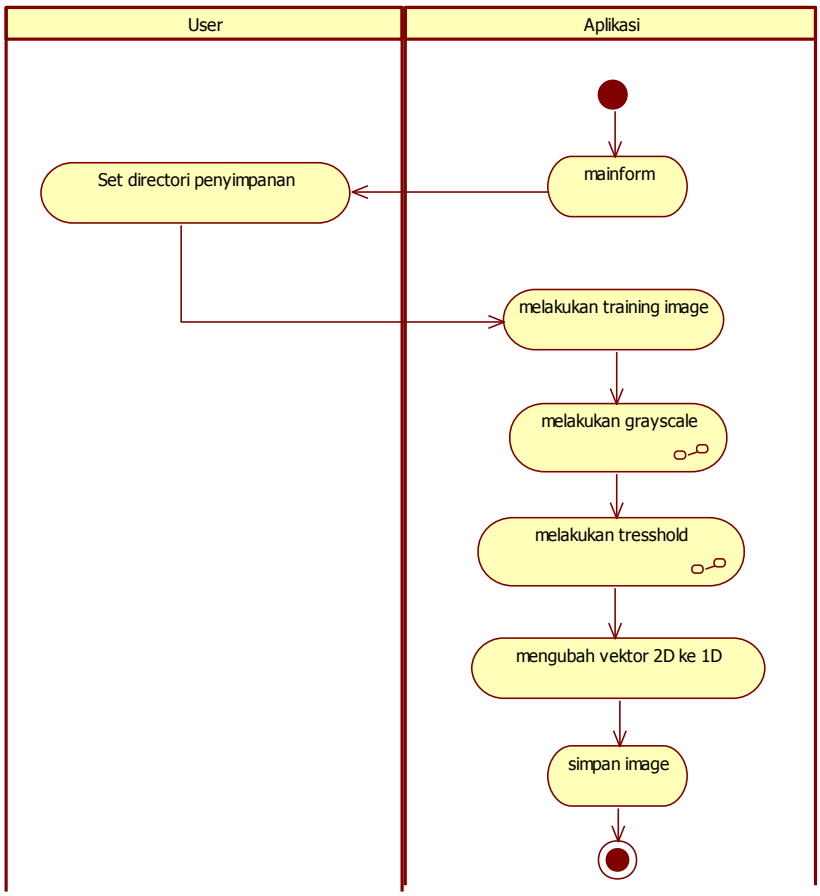
Fungsi ini melakukan pengambilan gambar wajah atau *capturing* untuk untuk dilakukan untuk *training image*.

Tabel 3.12 Skenario *Training Image*

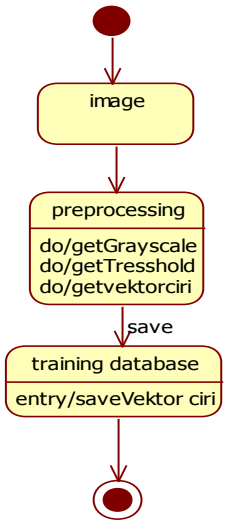
Identifikasi		
Nomor	2	
Nama	*Training image*	
Tujuan	Mengambil citra wajah dengan media *webcam* dan melakukan *training* terhadap citra wajah.	
Deskripsi	Pengguna mengambil citra wajah dengan *webcam* dengan jarak dan keadaan lingkungan yang telah ditentukan dan melakukan *training* terhadap citra wajah.	
Aktor	Pengguna	
Skenario Utama		
Kondisi awal	*Directory* telah di-*set* dan citra wajah telah diambil	
Aksi Aktor		**Reaksi Sistem**
Pengguna meng-*capture* citra wajah yang akan digunakan sebagai citra uji.		Sistem menangkap citra wajah dari media *scanning webcam.*
Kondisi akhir	*File image* berhasil di-*capture* dan di-*training*	



Gambar 3.21 *Sequence diagram training image*



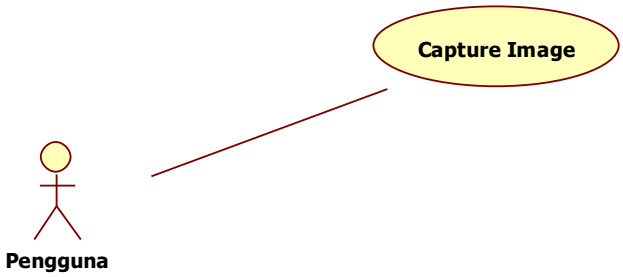
Gambar 3.22 ***Activity diagram training image***



Gambar 3.23 ***Statechart training image***

3.6.5 *Use Case Diagram, Skenario, Activity Diagram, Sequence Diagram, Statechart Diagram capture image*

Skenario proses-proses yang terjadi pada aplikasi, dipaparkan dengan tujuan untuk lebih memudahkan dalam pembangunan aplikasi, agar lebih konsisten.

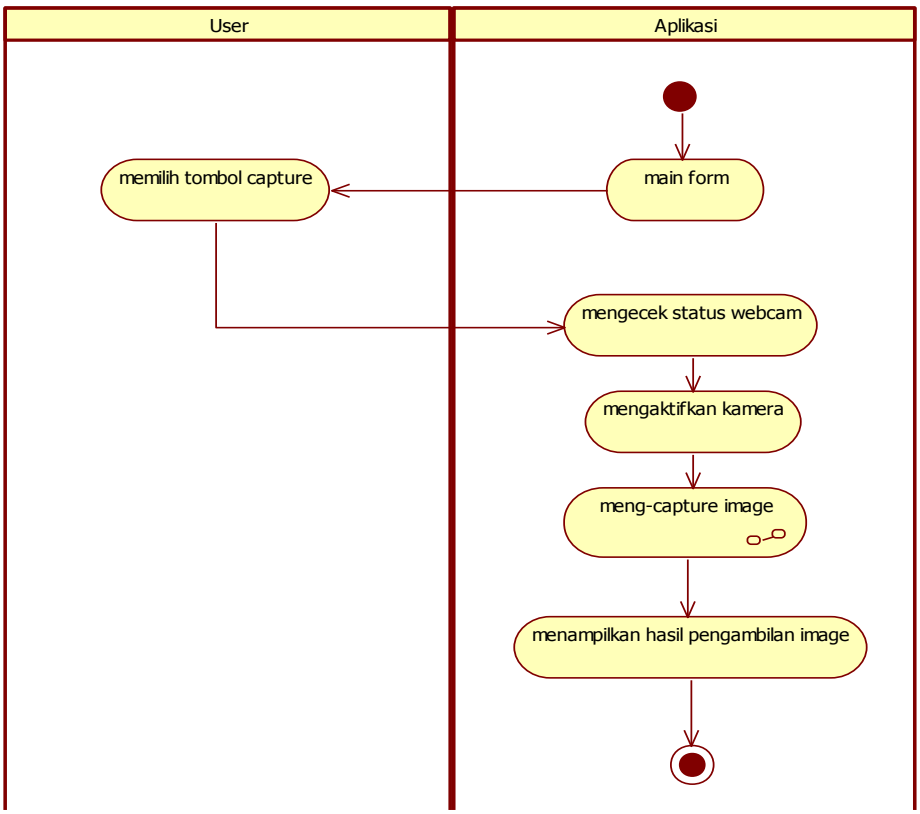


Gambar 3.24 *Use case Capture Webcam*

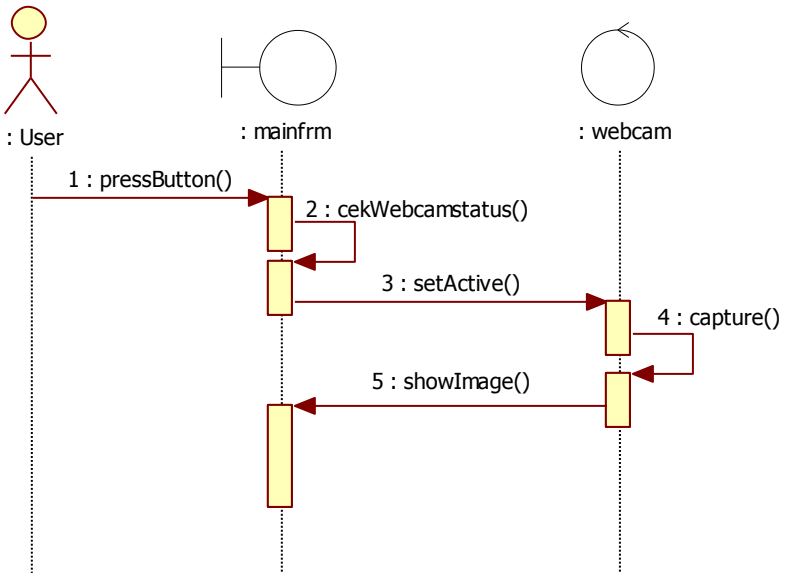
Fungsi ini melakukan pengambilan gambar wajah atau *capturing* untuk disimpan ke dalam *database* dan selanjutnya digunakan untuk proses *training image* maupun deteksi.

Tabel 3.13 Skenario Capture Image

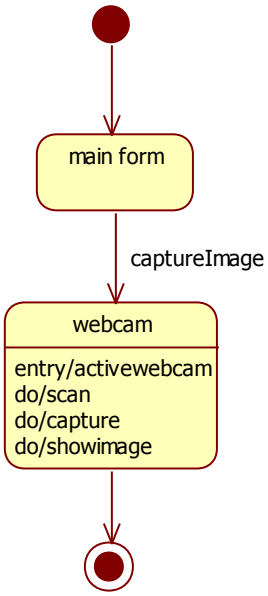
Identifikasi		
Nomor	3	
Nama	*Capture image*	
Tujuan	Mengambil citra wajah dengan media *webcam*	
Deskripsi	Pengguna mengambil citra wajah dengan *webcam* dengan jarak dan keadaan lingkungan yang telah ditentukan.	
Aktor	Pengguna	
Skenario Utama		
Kondisi awal	*Directory* telah di-*set*	
Aksi Aktor		**Reaksi Sistem**
Pengguna meng-*capture* citra wajah yang akan digunakan sebagai citra uji.		Sistem menangkap citra wajah dari media *scanning webcam.*
Kondisi akhir	*File image* berhasil di-*capture* dan ditampilkan.	



Gambar 3.25 ***Activity Diagram Proses Capture Image***

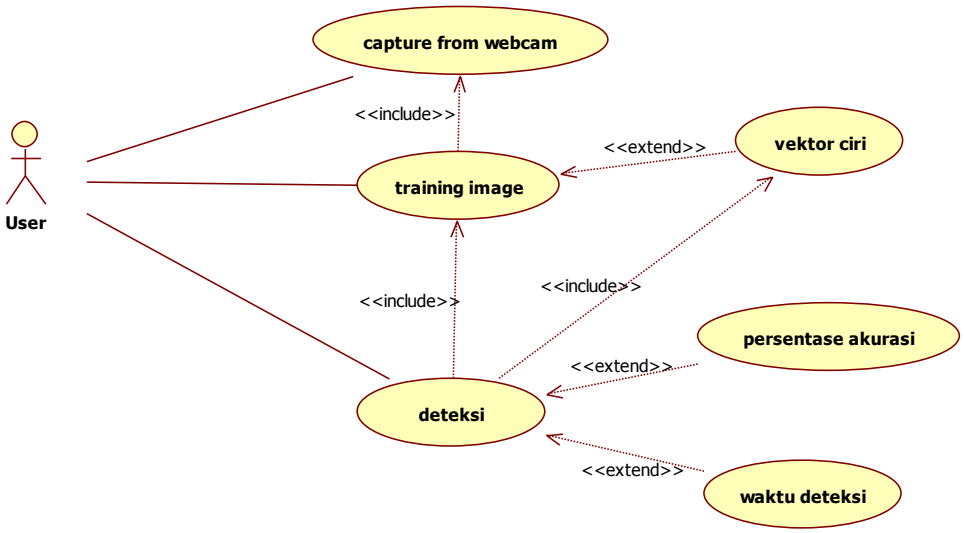


Gambar 3.26 ***Sequence diagram capture image***



Gambar 3.27 ***Statechart capture image***

3.6.6 *Use Case Diagram, Skenario, Activity Diagram, Sequence Diagram, Statechart Diagram deteksi*

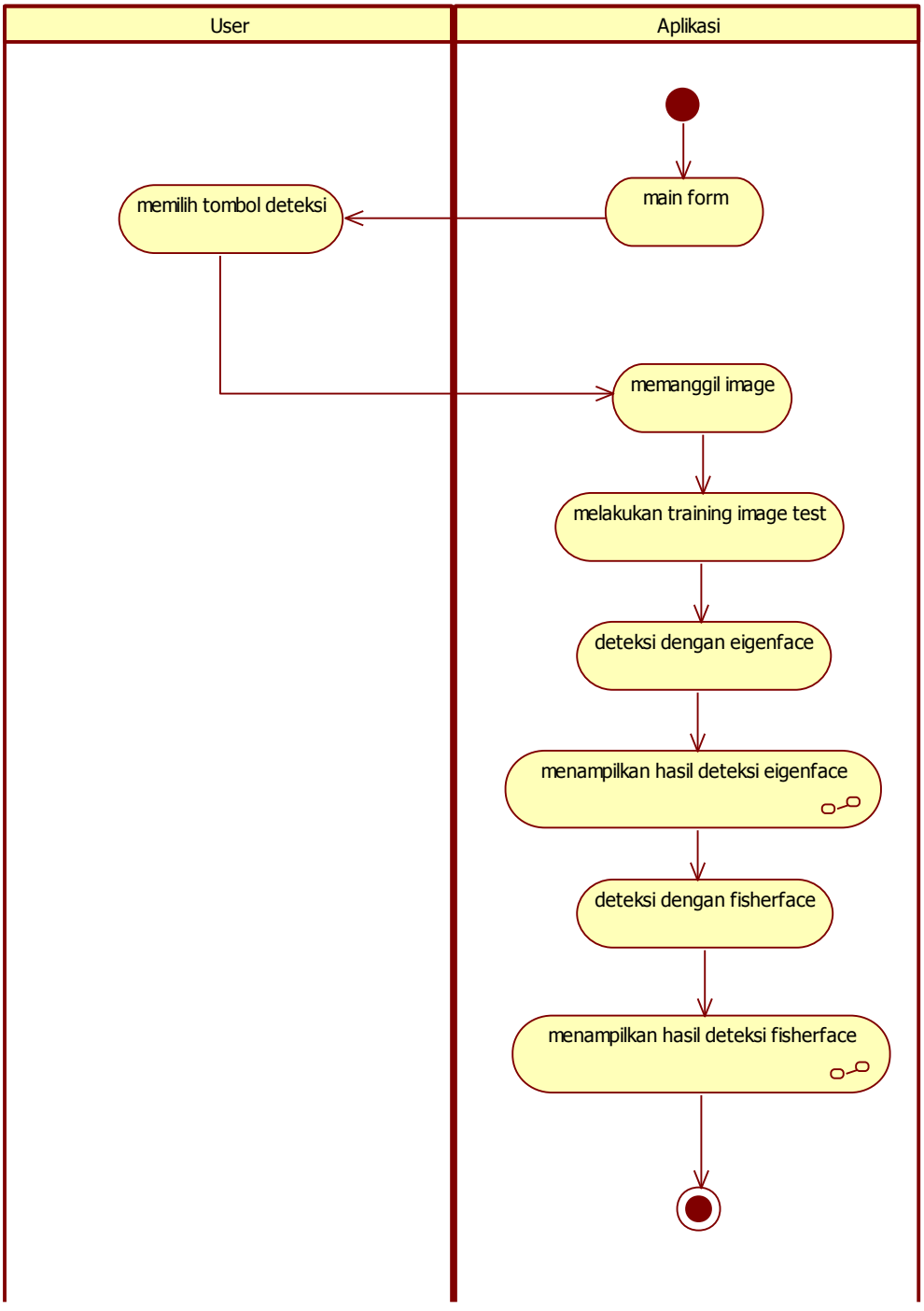


Gambar 3.28 ***Use case diagram proses deteksi***

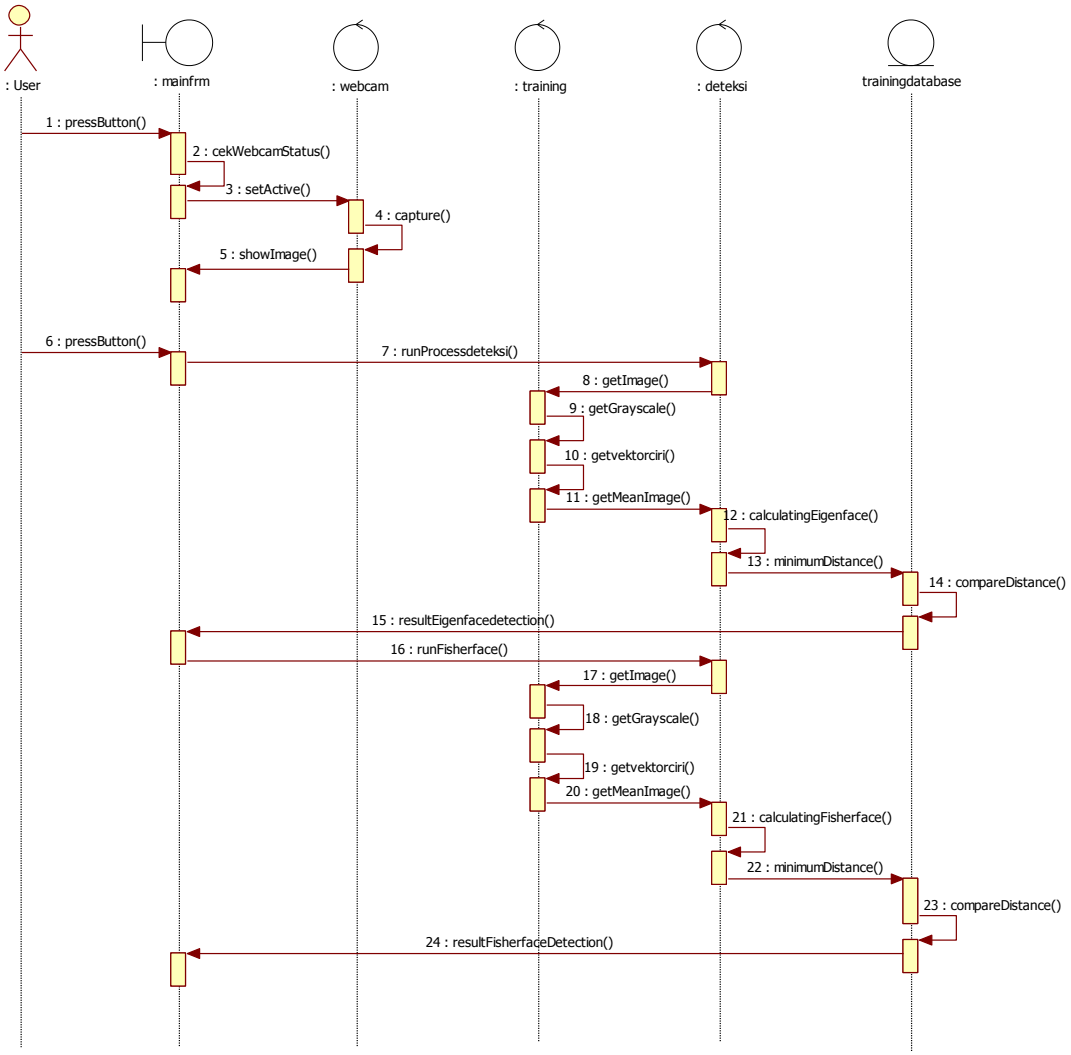
Fungsi ini melakukan deteksi perbandingan terhadap citra wajah hasil *capture* dengan vektor ciri yang telah dilakukan pada proses *training* yang disimpan di dalam *database.*

Tabel 3.14 Skenario proses deteksi

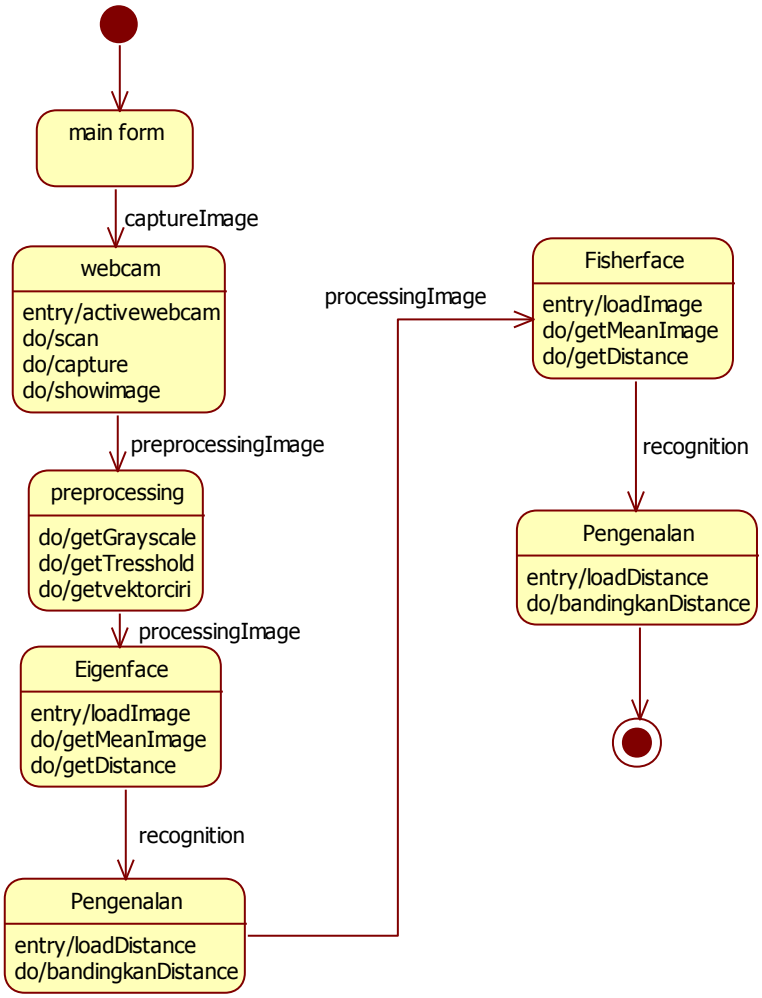
Identifikasi		
Nomor	4	
Nama	Deteksi	
Tujuan	Mencocokan citra wajah hasil *capture* dengan citra di *database*	
Deskripsi	Pengguna melakukan *capture* kembali dan melakukan deteksi untuk mencocokan dengan citra wajah yang telah tersimpan.	
Aktor	Pengguna	
Skenario Utama		
Kondisi awal	*Directory* telah di-*set* dan *training image* telah dilakukan.	
Aksi Aktor		**Reaksi Sistem**
Pengguna melakukan *capture* kembali dan melakukan deteksi untuk mencocokan dengan vektor ciri wajah yang telah tersimpan.		Sistem menangkap citra wajah dari pengguna dan melakukan pencocokan dengan vektor ciri wajah yang telah tersimpan di *database.*
Kondisi akhir	*File image* berhasil di-*capture* dan berhasil dikenali.	



Gambar 3.29 ***Activity diagram proses deteksi***



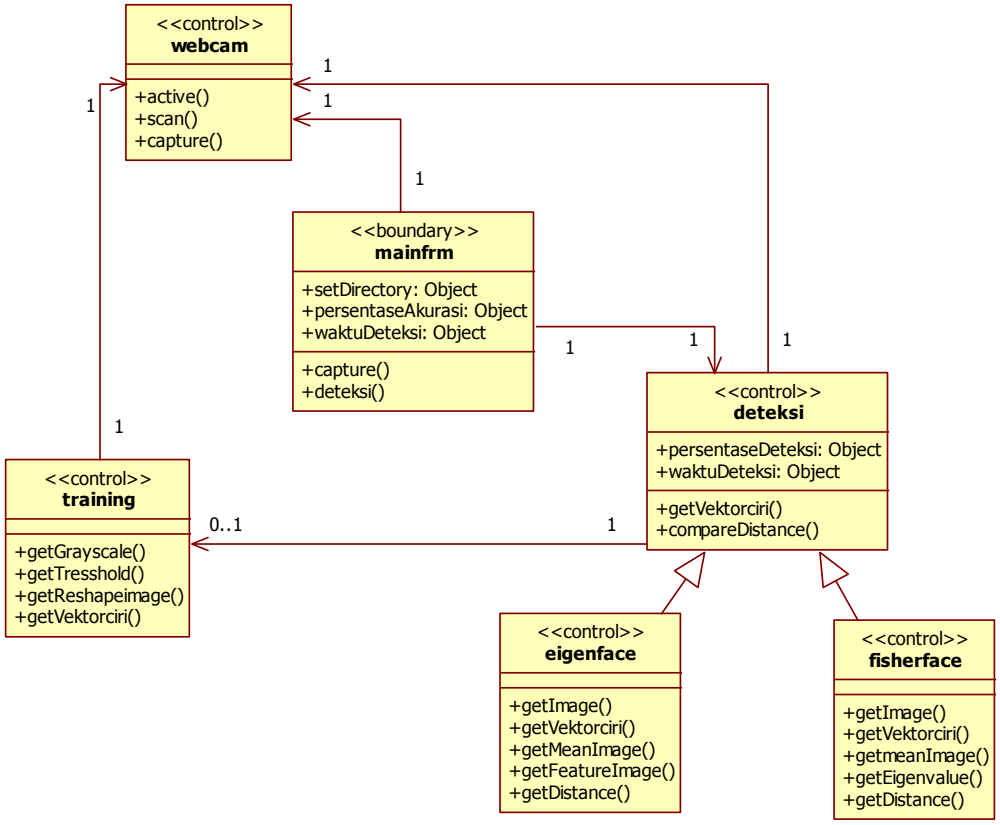
Gambar 3.30 ***Sequence diagram proses deteksi***



Gambar 3.31 ***Statechart proses deteksi***

3.6.7 *Class Diagram*

Class diagram menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metode/fungsi).



Gambar 3.32 *Clas Diagram* Aplikasi Perbandingan Algoritma

3.7 Perancangan Sistem

Perancangan merupakan bagian dari metodologi pembangunan suatu perangkat lunak yang harus dilakukan setelah melalui tahapan analisis. Pada bagian ini akan dijelaskan perancangan sistem yang dimaksudkan untuk menggambarkan sistem yang dibangun.

Perbandingan proses dan hasil pada sistem pengenalan pola wajah yang dibuat harus memiliki fungsi sebagai pembanding antara metode *Eigenface* dengan metode *Fisherface* untuk digunakan sebagai acuan untuk membentuk kesimpulan dan rekomendasi mengenai metode mana yang efektif dalam sistem pengenalan pola wajah yang dapat digunakan dalam keadaan yang telah dikondisikan sebelumnya,

Selain itu untuk membuktikan apakah kedua metode tersebut mempunyai performa dan kemampuan yang sesuai dengan kebutuhan apabila diimplementasikan pada sistem pengenalan pola wajah.

Saat awal aplikasi diaktifkan hal yang pertama yang akan dilakukan adalah melakukan pengaturan direktori yang akan digunakan untuk menyimpan *image* hasil *capture*, setelah itu kemudian melakukan proses pengambilan gambar yang dilakukan secara terus menerus dengan menggunakan perangkat *webcam*, gambar yang diperoleh satu persatu akan di-*capture*, yang kemudian secara otomatis akan diproses sekaligus dengan menggunakan metode *Eigenface* dan metode *Fisherface*.

Pada proses *training*, hasil *capture* (citra) akan diproses untuk mendapatkan vektor ciri, jika sudah didapatkan maka akan langsung disimpan ke dalam

database. Sedangkan pada proses *testing*, hasil *capture* akan diproses untuk dikenali pola wajah dengan dibandingkan dengan vektor ciri dari *image* yang sudah didapatkan dari proses *training* yang tersimpan di *database*.

Langkah-langkah yang dilakukan dalam tahapan perancangan sistem adalah sebagai berikut:

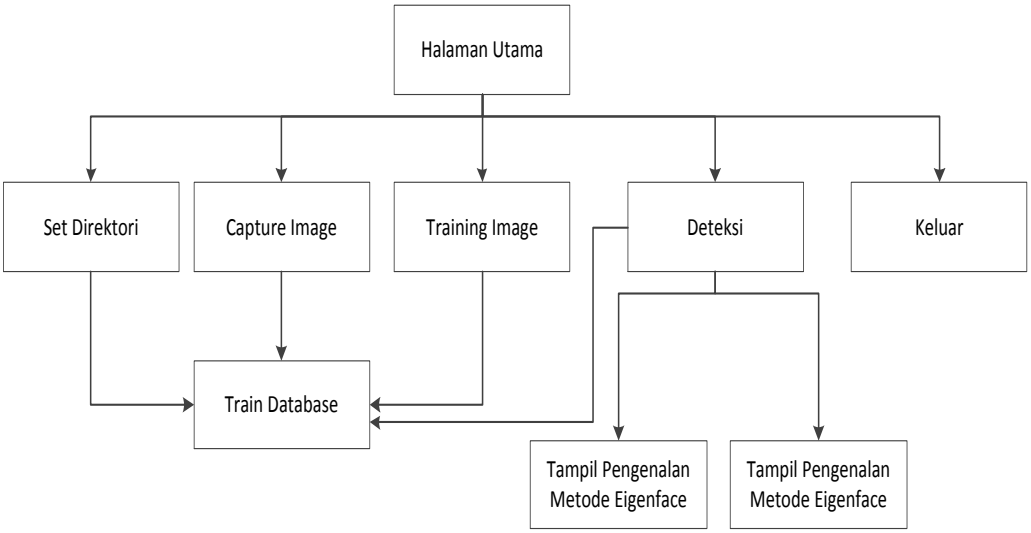
1. Perancangan Arsitektur
    a. Perancangan Struktur Menu
2. Perancangan Antarmuka
    a. Perancangan *Form*
    b. Perancangan Proses

3.7.1 Perancangan Arsitektur

Setelah melakukan perancangan data pada aplikasi yang dibangun, maka dilakukanlah perancangan arsitektur. Arsitektur disini dimaksudkan sebagai perancangan struktur menu yang akan dimiliki oleh sistem. Arsitektur sistem yang digunakan dalam aplikasi yang dibangun adalah arsitektur hirarki.

3.7.2 Perancangan Struktur Menu

Struktur menu dari aplikasi yang dibangun dapat dilihat pada (Gambar 3.33) dibawah ini :



Gambar 3.33 Rancangan Struktur Menu Aplikasi

3.7.3 Perancangan Antar Muka

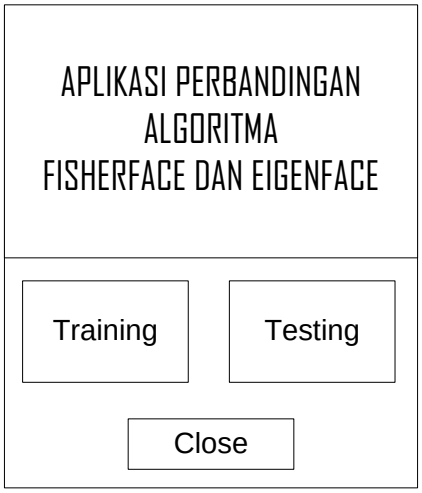
Perancangan antar muka merupakan perancangan yang dibuat sebelum program aplikasi dibuat, dimana perancangan ini bertujuan untuk menggambarkan tampilan program secara umum, yang nantinya diharapkan dapat memudahkan proses pembuatan aplikasi.

3.7.4 Perancangan Form

Perancangan *form-form* yang terdapat dalam aplikasi perbandingan hasil dan proses metode Eigenface dan metode Fisherface pada sistem pengenalan pola wajah, adalah sebagai berikut :

3.7.4.1 Perancangan *Form* Utama

Perancangan *form* utama dari aplikasi dapat dilihat pada (Gambar 3.34) dibawah ini :



Gambar 3.34 Rancangan Antar Muka Form Utama

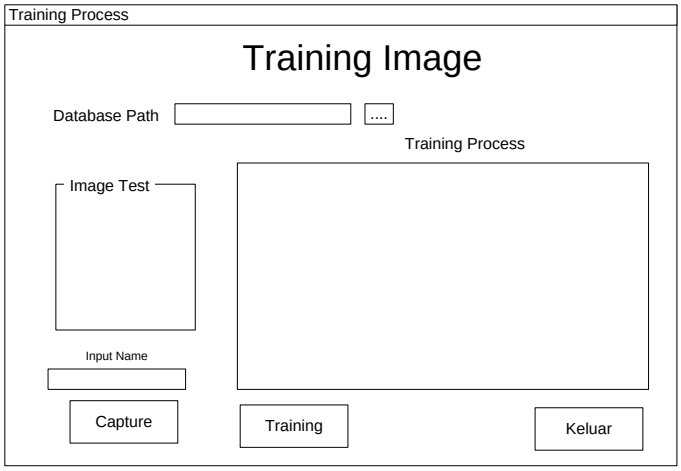
Dari *form* ini terdapat tiga buah tombol yaitu tombol *training*, tombol testing dan tombol untuk keluar dari aplikasi. Deskripsi objek dari *form* utama dapat dilihat pada Tabel 3.15 dibawah ini :

Tabel 3.15 Deskripsi Objek *Form* utama

Objek	Property	Setting
Button1	Name Caption	btTraining
Button2	Name Caption	btTesting
Button3	Name Caption	btCloce

3.7.4.2 Perancangan Form Training

Perancangan form *training image* dapat dilihat pada (Gambar 3.35) dibawah ini :



Gambar 3.35 Rancangan Antar Muka Form Training

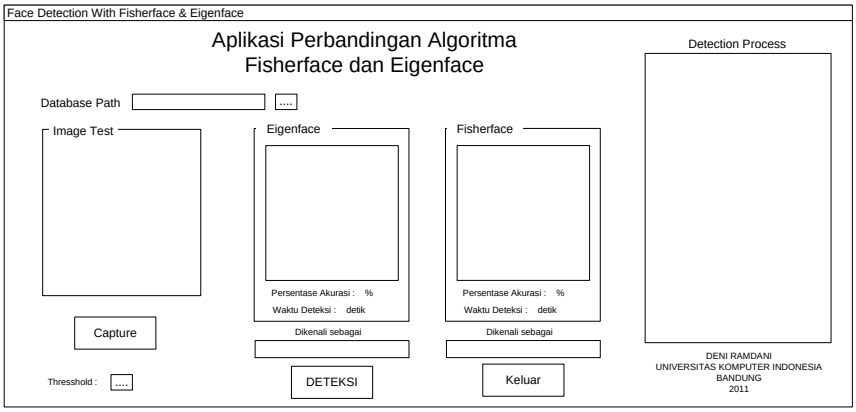
Dari *form* utama, ketika tombol *training* yang dipilih. Maka akan muncul jendela *training image* yang digunakan untuk melakukan pengambilan pola dari *training set* yang akan digunakan pada proses pengenalan. Deskripsi objek dari *form* utama dapat dilihat pada Tabel 3.16 dibawah ini :

Tabel 3.16 Deskripsi Objek Form Training

Objek	Property	Setting
Label1	Caption	Database Path
Label2	Caption	Training Process
Label3	Caption	Input Name
Text1	Name	edTrain1
Text2	Name	edGambar1
axes	Name Caption	axes1
Button1	Name Caption	btTrain
Button2	Name Caption	btCapture
Button3	Name Caption	btTraining
Button4	Name Caption	btKeluar

3.7.4.3 Perancangan Form Deteksi

Perancangan form deteksi atau pengenalan dapat dilihat pada (Gambar 3.36) dibawah ini :



Gambar 3.36 Tampilan Form *Training Image*

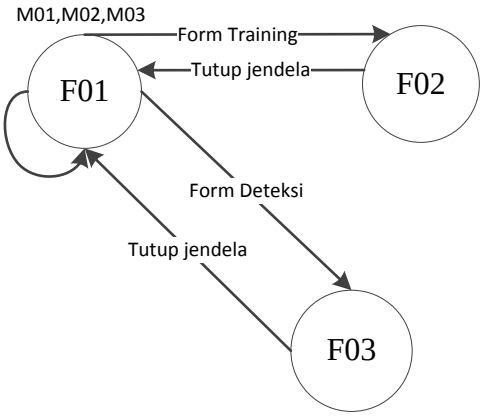
Dari *form* utama, setelah proses *training image* selanjutnya adalah proses pengenalan. Maka akan muncul jendela deteksi yang digunakan untuk melakukan pengenalan terhadap *image* uji. Deskripsi objek dari *form* utama dapat dilihat pada Tabel 3.17 dibawah ini :

Tabel 3.17 Deskripsi Objek Form Deteksi

Objek	Property	Setting
Label1	Caption	Database Path
Label2	Caption	DetectionProcess
Label3	Caption	Input Name
Label4	Caption	Thresshold
Text1	Name	edTrain1
Text2	Name	edGambar1
Text3	Name	edGambar2
Text4	Name	edTH
Button1	Name Caption	btTrain
Button2	Name Caption	btCapture
Button3	Name Caption	btDeteksi
Button4	Name Caption	btKeluar

3.7.5 Perancangan Jaringan Semantik

Jaringan semantik menggambarkan keterhubungan navigasi menu dari satu menu ke menu yang lain di dalam sebuah aplikasi atau sistem.



Gambar 3.37 Jaringan Semantik Aplikasi