BAB 2

TINJAUAN PUSTAKA

2.1 Citra Digital

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

1. Optik, berupa foto.
2. Analog berupa sinyal video, seperti gambar pada monitor televisi.
3. Digital yang dapat langsung disimpan pada suatu pita magnetik.

Citra dapat dikelompokkan menjadi dua bagian yaitu citra diam (*still image*) adalah citra tunggal yang tidak bergerak dan citra bergerak (*moving image*) yaitu rangkaian citra diam yang ditampilkan secara beruntun (*sekuensial*), sehingga memberi kesan pada mata sebagai gambar yang bergerak. Setiap citra didalam rangkaian itu disebut *frame*. Gambar-gambar yang tampak pada film layar lebar atau televisi yaitu terdiri dari ratusan sampai ribuan *frame*. [1]

Dari sudut pandang pencitraan, citra (*image*) adalah rekaman hasil interaksi antara gelombang dengan benda (*object*), yang memberikan sebagian gambaran atau informasi dari benda tersebut. Proses pembentukan citra dengan merekam hasil interaksi inilah yang disebut sebagai proses pencitraan (*imaging*). Dengan demikian ada 3 (tiga) komponen utama dalam pencitraan, yaitu :

1. Gelombang pengindera (*sensing waves*)
2. Benda (*object*)
3. Alat pengindera (*sensor*)

Untuk sebuah sistem koordinat merah-hijau-biru, nilai instanious trimulusnya adalah:

$$R(x,y,t) = C(x,y,t,\lambda)Rs(\lambda)d\lambda \int_{\infty 0}$$
$$G(x,y,t) = C(x,y,t,\lambda)Gs(\lambda)d\lambda \int_{\infty 0}$$
$$B(x,y,t) = C(x,y,t,\lambda)Bs(\lambda)d\lambda \int_{\infty 0}$$

Jika Rs(λ),Gs(λ),Bs(λ) adalah nilai spectral trimulusnya untuk himpunan warna primer merah, hijau dan biru, maka spectral trimulus adalah dalam efek, nilai trimulus dibutuhkan untuk memperlihatkan sejumlah cahaya dengan panjang gelombang λ. Dalam sebuah sistem citra multispectral, *field* citra diamati dan dimodelkan sebagai sebuah integral berat *spectral* dari fungsi cahaya citra. *Field spectral* citra adalah:

$$F_1(x,y,t) = C(x,y,t,\lambda)Rs(\lambda)d\lambda \int_{\infty 0}$$

Di mana $S_1(\lambda)$ adalah respon *spectral sensor*.

2.1.1 Pengolahan Citra

Pengolahan citra digital adalah pemrosesan citra menjadi citra yang lain dengan kualitas yang lebih baik, yaitu pemrosesan pada usaha untuk memanipulasi [9]. Citra yang telah menjadi gambar lain menggunakan algoritma atau teknik tertentu. Pengolahan citra mempunyai tujuan yaitu:

1. Proses memperbaiki kualitas citra agar mudah diinterpretasikan oleh manusia atau komputer.
2. Teknik pengolahan citra dengan mentrasformasikan citra menjadi citra lain.
3. Pengolahan citra merupakan proses awal dari komputer vision.

Pada umumnya pengolahan citra berhubungan dengan citra-citra digital. Dalam hal ini, citra f(x,y) diperoleh secara diskrit dan kemudian dikuantisasi. Maka akan diperoleh suatu citra baru : (m,n) → I dengan m,n ∈ I; di mana I adalah himpunan bilangan bulat (*integer*). Namun demikian, secara umum sistem pengolahan citra mengandalkan citra asal yang bernilai *real* dan menghasilkan bilangan *real* juga, meskipun secara teknis pada akhirnya citra ini didigitalkan sebelum disimpan.

Ada beberapa hal yang penting didalam pengolahan citra digital, antara lain teknik-teknik pengambilan citra, model citra digital, sampling dan kuantitasi, histogram, proses *filtering*, perbaikan citra sampai pada pengolahan citra digital yang lebih lanjut seperti segmentasi, *image clustering* dan ekstrasi ciri.[1]

2.1.2 Pengertian Citra Digital

Citra digital dapat diartikan sebagai suatu fungsi kontinu dari intensitas cahaya dalam bidang dua dimensi. Pada dasarnya suatu citra terdiri dari sekumpulan piksel, yang tiap pikselnya terdiri dari tiga kombinasi warna dasar yaitu merah, hijau, biru, (RGB), dengan intensitas yang berbeda-beda.

Citra digital merupakan representatif dari citra yang diambil oleh mesin dengan bentuk pendekatan berdasarkan sampling dan kuantitasi. Sampling menyatakan besarnya kota-kotak yang disusun dalam baris dan kolom. Dengan kata lain sampling pada citra menyatakan besar kecilnya ukuran *pixel* (titik) pada citra, jika kuantitasi menyatakan besarnya nilai tingkat kecerahan yang dinyatakan dalam nilai tingkat keabuan (*gray scale*) sesuai dengan jumlah bit biner yang digunakan oleh mesin, maka dapat dikatakan bahwa kuantitasi pada citra menyatakan jumlah warna yang ada pada citra.[3]

Secara umum teknik pengolahan citra digital dibagi menjadi 3 tingkat pengolahan, yaitu:

1. Tahap 1 yang dinamakan dengan *Low-Level Processing* (pengolahan tingkat rendah). Pengolahan ini operasional-operasional dasar dalam pengolahan citra, seperti pengurangan *noise* (*noise redution*), perbaikan citra (*image enhancement*) dan restorasi citra (*image restoration*).
2. Tahap 2 yang dinamakan dengan *Mid-Level Processing* (pengolahan tingkat menengah). Pengolahan ini meliputi segmentasi pada citra, deskripsi objek dan klasifikasi objek secara terpisah.
3. Tahap 3 yang dinamakan dengan *High-Level Processing* (pengolahan tingkat tinggi), yang meliputi analisis citra.

Operasi–operasi pengolahan citra meliputi perbaikan kualitas citra, yakni perbaikan kualitas citra ini bertujuan memperbaiki kualitas citra dengan memanipulasi parameter-parameter citra. Melalui operasi ini, ciri-ciri khusus yang terdapat dalam didalam citra dapat lebih ditonjolkan. Dalam perbaikan kualitas

citra dapat dilakukan operasi–operasi citra, seperti yang tertulis dalam buku Pengolahan Citra Digital yaitu :

a. Perbaikan kontras gelap dan terang
b. Perbaikan tepian objek
c. Penajaman
d. Pemberian warna semu
e. Penapisan derau

Gambar adalah contoh operasi penajaman. Operasi ini menerima masukan sebuah citra yang gambarnya hendak dibuat tampak lebih tajam. Bagian citra yang ditajamkan adalah tepi-tepi objek.

2.2 Pengenalan Wajah *(Face Recognition)*

Pengenalan wajah adalah suatu metode pengenalan yang berorientasi pada wajah. Pengenalan ini dapat dibagi menjadi dua bagian yaitu : Dikenali atau tidak dikenali, setelah dilakukan perbandingan dengan pola yang sebelumnya disimpan di dalam *database*. Metode ini juga harus mampu mengenali objek bukan wajah. Perhitungan model pengenalan wajah memiliki beberapa masalah. Kesulitan muncul ketika wajah direpresentasikan dalam suatu pola yang berisi informasi unik yang membedakan dengan wajah yang lain.[4]

Secara umum sistem pengenalan citra wajah dibagi menjadi dua jenis, yaitu sistem *feature based* dan sistem *image based*. Pada sistem pertama digunakan fitur yang diekstraksi dari komponen citra wajah (mata, hidung, mulut, dll) yang kemudian hubungan antar fitur-fitur tersebut dimodelkan secara geometris.

Sedangkan sistem kedua menggunakan informasi mentah dari piksel citra yang kemudian direpresentasikan dalam metode tertentu.

Proses pengenalan wajah yang dilakukan oleh komputer tidak semudah dan secepat proses pengenalan yang dilakukan oleh manusia. Manusia dapat dengan mudah mengenali wajah seseorang dengan cepat tanpa rasa harus berfikir. Manusia juga tidak terpengaruh oleh orientasi wajah tersebut, misalnya wajah orang tersebut dengan keadaan agak menoleh, merunduk dan menengadah asalkan ada batas-batas yang masih bias dilihat. Sedangkan komputer selain lamban dalam pengenalan juga kesulitan pada orientasi wajah berlainan, pencahayaan, latar belakang yang berbeda, potongan rambut, kumis atau jenggot, berkacamata atau tidak dan sebagainya.

Memang otak manusia lebih memiliki keunggulan dalam mengatasi masalah dimana aturan eksplisit tidak dapat dengan mudah diformulasikan, sedangkan komputer mempunyai keuntungan pada bidang seperti matematika dimana aturan-aturan mudah diformulasikan. Oleh karena itu banyak dilakukan penelitian untuk mencari algoritma-algoritma yang tepat bagi komputer agar dapat mengenali suatu wajah dengan memperhatikan faktor kecepatan dan akurasi.

Metode pengenalan pola wajah memakai dua prosedur yaitu :

a. Pengenalan kontur wajah dengan mengenali bentuk hidung, mata dan mulut dan korelasi diantara keduanya. Karakteristik organ tersebut kemudian dinyatakan dalam bentuk vektor.
b. Analisis komponen wajah, berdasarkan informasi dari konsep ini, mencari perhitungan model terbaik yang menjelaskan bentuk wajah

dengan mengutip informasi yang relevan yang terkandung di dalam wajah tersebut.

Dibalik kemudahan mengenali wajah, ada bebrapa masalah yang mungkin timbul dalam proses pengenalan wajah, yaitu : adanya perubahan skala, perubahan posisi, perubahan posisi, perubahan pencahayaan atau adanya perubahan detail dan ekspresi wajah.

Alan Brooks pernah mengembangkan sebuah penelitian yang membandingkan dua algoritma yaitu *Eigenface* dan *Fisherface*. Penelitian ini difokuskan pada perubahan pose wajah apakah mempengaruhi akurasi pengenalan wajah. Diberikan *database* latih berupa foto wajah manusia, kemudian digunakan untuk melatih sebuah sistem pengenalan wajah, setelah proses latihan selesai, diberikan sebuah masukan *image* yang sebenarnya sama dengan salah satu *image* wajah pada *fase* latihan tetapi dengan pose yang berbeda. Sistem juga diharapkan punya sensitifitas minimal terhadap pencahayaan. Sistem dikembangkan dengan dua algoritma yaitu *Eigenface* dan *Fisherface*, dan dibandingkan hasilnya. Kedua teknik menghasilkan hasil yang memuaskan tetapi ada beberapa perbedaan Pada *Eigenface* kompleksitas komputasi lebih sederhana daripada *Fisherface*. Dari segi efektifitas karena perubahan pose *Fisherface* memberikan hasil yang lebih baik, bahkan dengan data yang lebih terbatas. Teknik *Eigenface* juga lebih sensitif terhadap pencahayaan dibandingkan dengan *Fisherface.*

2.2.1 Tahapan Umum Pengenalan Wajah

Algoritma untuk mengenali wajah ada bermacam-macam. Akan tetapi pada umumnya ada beberapa tahapan-tahapan umum yang dipakai dalam pengenalan wajah.[4]

Tahapan-tahapan umum dari pengenalan wajah adalah sebagai berikut :

1. Perbaikan Gambar
2. Segmentasi
3. Pencarian *Feature*
4. *Scew Detection/*Perbaikan Kemiringan
5. Identifikasi

2.2.1.1 Perbaikan Gambar

Masukkan *(input)* gambar dari hasil *scanning* belum tentu bagus atau kabur, sehingga bukan merupakan *input* yang bagus bagi program. *Image* hasil *scanning* biasanya mengandung banyak *noise* yang harus dihilangkan. Kemudian harus diterapkan algoritma-algoritma untuk mempertajam gambar bila gambar kabur. Seperti *filtering* yang teorinya ada pada *image processing*. Setelah itu dilakukan pengaturan pencahayaan paga *image*, sehingga diharapkan semua *image* mempunyai pencahayaan yang sama untuk memudahkan proses pengenalan nanti.

2.2.1.2 Segmentasi

Tahap ini dilakukan segmentasi dengan satu algoritma untuk mengenali gambar mana yang merupakan wajah. Apabila gambar setelah dicek oleh

algoritma bukan merupakan wajah, maka proses berhenti dan keluar. Akan tetapi bila wajah tersebut ada bentuk oval dari wajah, maka *image* tersebut akan disegmentasi untuk membuang latar belakangnya sehingga hasilnya adalah wajah oval saja tanpa latar belakang. Yang dimaksud wajah oval yaitu bentuk oval dari wajah, yaitu wajah tanpa rambut, telinga atau leher.

2.2.1.3 Pencarian *Feature*

Ada dua macam *feature* pada wajah, yaitu *holistic feature* dan *partian feature* pada pengenalan wajah biasanya disebut sebagai *facial feature*. Contoh *facial feature*-nya adalah warna dan bentuk rambut, besar dan letak mata, hidung, rambut, mulut, telinga dan lain-lain. Sedangkan pada *holistic feature*, setiap *feature*-nya adalah suatu karakteristik dari seluruh wajah. Maksudnya wajah dianggap sebagai suatu kesatuan yang utuh. *Participal Component Analysis* adalah teknik yeng menggunakan *holistic feature*. Jadi *Participal Component Analysis* menganggap suatu wajah adalah suatu kesatuan dan dibandingkan dengan wajah yang lain.

2.2.1.4 Perbaikan Kemiringan

Gambar oval dari wajah seseorang didapatkan dari hasil segmentasi dapat tegak, miring ke kiri atau ke kanan atau bahkan terbalik. Oleh karena itu proses perbaikan kemiringan ini diperlukan, sehingga gambar yang miring atau terbalik bisa diperbaiki agar menjadi gambar yang benar-benar tegak.

Sebenarnya pada gambar wajah, kemiringan bukan hanya sesederhana bila kita mengamati tulisan. Tulisan atau huruf hanya bisa miring ke kiri atau ke kanan. Pada gambar selain miring ke kiri atau ke kanan, ada juga kemiringan pada orientasi wajah. Maksudnya gambar orang tidak selalu menghadap ke depan, namun bisa menoleh beberapa derajat ke kanan atau ke kiri. Maka seharusnya ada algoritma untuk mengatur orientasi wajah tersebut.

Facial Feature biasanya dimanfaatkan untuk perbaikan kemiringan. Misalkan dengan mengamati keadaan lurus atau miringnya dari letak dua mata, maka dapat ditentukan pada sudut rotasi bagi wajah. Letak hidung juga harus diperhatikan agar rotasi wajah yang dihasilakan tidak terbalik., sehingga mata harus diatas hidung dan bukan sebaliknya. Berdasarkan *feature* ini maka gambar hanya bisa dirotasi kemiringannya sehingga menjadi tegak, tetapi tetap saja tidak ada algoritma untuk orientasi wajah.

2.2.1.5 Identifikasi

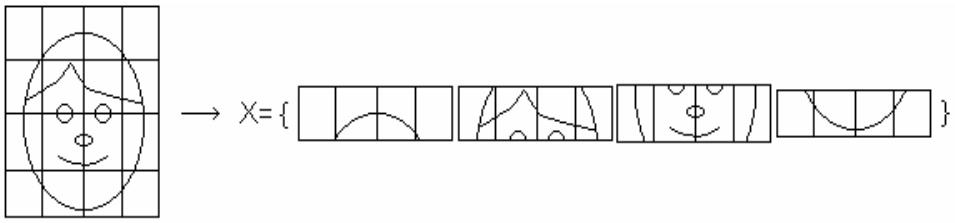
Pada tahap akhir akan diambil kesimpulan. Kesimpulan yang diambil bisa berbeda tergantung pada aplikasi yang diinginkan, misalnya kesimpulan yang diambil yaitu untuk menggolongkan individu pada *image* berdasarkan jenis kelaminnya, atau melakukan pengenalan terhadap *image* yang di *input*-kan atau langsung memberi identitas *image* tersebut.

Setelah komputer atau tentang *feature* dan *testing image* yang di *input*-kan maka dapat dilakukan perbandingan dengan *feature-feature image* pelatihan. Tentu saja sebelumnya *image-image* pelatihan juga melalui tahap yang sama dari

tahap satu sampai lima. Setelah dilakukan perbandingan maka dipilih satu wajah *training* yang *feature*-nya paling mirip dengan *image testing.* Wajah *training* inilah yang merupakan *output* dari pengenalan wajah, yaitu sebagai wajah pelatihan yang mirip dengan testing.

2.2.2 Vektor dan Ruang Wajah

Sebuah wajah, yang juga merupakan sebuah citra, dapat dipandang sebagai sebuah vektor. Apabila **lebar** dan **tinggi** citra adalah **w** dan **h** piksel, maka



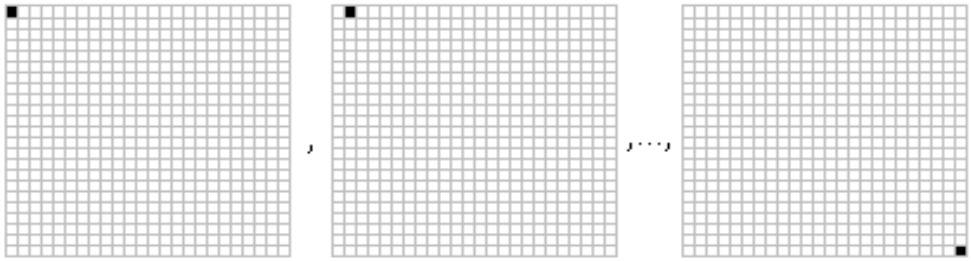
Gambar 2.1 Formasi Vektor dari Sebuah Gambar

banyaknya komponen dari vektor ini adalah **(w x h)**. Setiap piksel dikodekan oleh satu komponen vektor. Konstruksi vektor dari sebuah citra dibentuk oleh penggabungan sederhana, yaitu baris dari sebuah citra diletakkan saling bersebelahan dengan baris-baris yang lain, seperti yang terlihat pada gambar 2.1.

Vektor wajah yang telah dideskripsikan sebelumnya merupakan bagian dari sebuah ruang. Ruang ini adalah sebuah ruang lingkup citra (citra *space*), yaitu ruang dari semua citra keseluruhan yang mempunyai dimensi **w*x*h** piksel. Basis dari ruang lingkup citra dikomposisikan oleh vektor-vektor berikut, seperti terlihat pada gambar 2.2. Semua wajah mirip satu sama lain. Mereka semua mempunyai dua mata, satu hidung, satu mulut, dua telinga, dan lain sebagainya yang terletak pada tempat yang sama. Akibatnya semua vektor wajah terletak

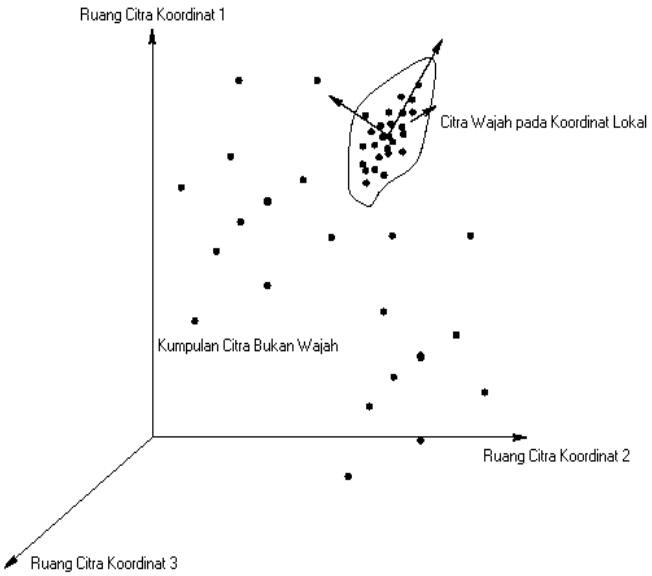
pada tempat-tempat yang amat berdekatan dalam ruang lingkup citra. Oleh karena itu, sebuah ruang citra (citra *space*) yang penuh bukan sebuah ruang yang tepat untuk mendeskripsikan sebuah wajah. Penelitian yang akan dilakukan berikut bertujuan untuk membangun sebuah ruang lingkup wajah (*face space*) yang lebih dapat mendeskripsikan wajah. Vektor basis dari ruang lingkup wajah (*face space*) disebut sebagai komponen utama (*principal component*).

Dimensi dari ruang lingkup citra adalah **w*x*h**. Tentu saja penggunaan semua piksel dari sebuah wajah tidak relevan, dikarenakan setiap piksel memiliki kebergantungan dengan tetangganya. Sehingga dimensi dari ruang lingkup wajah akan lebih kecil jika dibandingkan dengan dimensi ruang lingkup citranya.



Gambar 2.2 Basis Ruang Lingkup Citra

Hubungan antara ruang citra yang berdimensi tinggi dan ruang wajah berdimensi rendah yang memuat komponen utama dapat digambarkan dengan diagram sebagai berikut.



Gambar 2.3 Ruang Citra dan Ruang Wajah

2.3 Algoritma Pengenalan Wajah

Sekarang pengenalan wajah telah dikembangkan untuk banyak aplikasi, terutama untuk aplikasi keamanan. Penggunaan wajah sebagai *identifier* mempunyai banyak manfaat, terutama kepraktisannya karena tidak memerlukan kartu atau foto untuk identifikasi. Masalah utamanya adalah sebuah *image* yang mewakili sebuah gambar yang terdiri dari *vector* yang berukuran relatif besar. Ada banyak teknik untuk mereduksi dimensi *image* yang diproses. Diataranya adalah menggunakan *eigenface algoritm* dan *fisherface algoritm.*

2.3.1 Metode *Eigenface*

Kata *eigenface* sebenarnya berasal dari bahasa Jerman *"eigenwert"* dimana *"eigen"* artinya karakteristik dan *"wert"* artinya nilai. *Eigenface* adalah salah satu algoritma pengenalan pola wajah yang berdasarkan pada *Principle Component*

Analysis (PCA) yang dikembangkan di MIT. *Eigenface* merupakan kumpulan dari *eigenvector* yang digunakan untuk masalah *computer vision* pada pengenalan wajah manusia.[3]

Eigenface adalah sekumpulan *standardize face ingredient* yang diambil dari analisis statistik dari banyak gambar wajah. Untuk menghasilkan *eigenface*, sekumpulan citra digital dari wajah manusia diambil pada kondisi pencahayaan yang sama kemudian dinormalisasikan dan diproses pada resolusi yang sama (misal **m x n**), kemudian citra tadi diperlakukan sebagai vektor dimensi **m x n** dimana komponennya diambil dari nilai piksel citra.

Teknik reduksi dimensi yang banyak digunakan dalam pengenalan pola, khususnya pengenalan wajah, adalah *Principal Component Analysis* (PCA) atau yang disebut juga transformasi *Karhunen-Loeve*. Teknik reduksi inilah yang menjadi dasar dari metode Eigenface [3].

Principal Components Analysis (PCA) merupakan suatu teknik yang digunakan untuk mereduksi data multidimensi menjadi data dengan dimensi yang lebih kecil, dengan tetap mempertahankan karakteristik dari data tersebut, untuk menganalisis data dan membuat model peramalan, dalam perhitungannya melibatkan nilai *eigen* dari matriks kovarians *(singular value decomposition).*

Secara matematis *Principal Components Analysis* (PCA) adalah sebuah transformasi linear orthogonal, yaitu transformasi yang memberikan nilai optimal yang didapat dengan metode kuadrat terkecil.[3]

Selain itu *Principal Components Analysis* (PCA) merupakan teknik yang popular digunakan dalam pengenalan wajah, walaupun bukan merupakan teknik

yang paling optimal, *Principal Components Analysis* (PCA) digunakan sebagai bahan alternatif dari penggunaan *Linear Discriminant Analysis* (LDA). *Principal Components Analysis* (PCA) menggunakan jarak L_2 *(Euclidian)* untuk meminimalkan kesalahan.

Dalam penggunaannya, *Principal Components Analysis* (PCA) menggunakan beberapa asumsi yaitu data yang linear, formula statistika yang digunakan hanya rata-rata dan kovarians, data dengan varians yang besar dianggap lebih penting.

Berdasarkan kesimpulan yang diberikan oleh *Kirby* dan *Sirovich* (1990). Ditunjukan bahwa basis optimal yang dapat digunakan untuk merepresentasikan suatu vektor citra diberikan oleh ruang *eigen* yang dibentuk dari nilai-nilai *eigen* yang tidak nol *(nonzero eigenvalues)* dari matriks kovarian seluruh citra tersebut.

Dengan menggunakan basis yang dibentuk oleh ruang *eigen* tersebut dapat dilakukan reduksi dimensi dengan melakukan transformasi linier dari suatu ruang berdimensi tinggi ke dalam ruang berdimensi lebih rendah. Untuk menentukan dimensi yang lebih rendah dengan galat *(error, information losss)* yang minimum dapat dilakukan dengan memilih sejumlah nilai eigen yang terbesar dari ruang berdimensi tinggi tersebut. Tahapan tersebut merupakan gambaran umum dari metode yang disebut Analisis Komponen Utama *(Principal Component Analysis)*.

Turk dan Pentland (1991) mengatakan bahwa jika ada *M* buah citra wajah yaitu $\Gamma_1, \Gamma_2, ...\Gamma_M$ yang digunakan dalam pelatihan dan masing-masing citra wajah dipandang sebagai sebuah vektor wajah yang panjangnya N (baris x

kolom). Rata-rata wajah dari kumpulan data yang dilatih ini dapat diperoleh dengan persamaan (2.1) berikut ini :

$$\Psi = \sum_{i-1}^{M} \Gamma_i \qquad (2.1)$$

Kemudian, setiap vektor wajah dikurangkan dengan rata-rata wajah, dengan persamaan (2.2).

$$\Phi = \Gamma_i - \Psi \qquad (2.2)$$

Kumpulan dari vektor-vektor yang sangat besar ini merupakan syarat untuk analisis komponen utama, yang mana mencari sekumpulan M (M<<N) buah vektor orthonormal μ_k yang bersesuaian dengan nilai eigen λ_k terbaik yang menggambarkan pendistribusian data. Dimana kumpulan dari vektor *eigen* dengan nilai eigen terbaik ini merupakan vektor basis dari PCA atau biasa disebut dengan *Eigenface*. Vektor μ_k dan λ_k skalar masing-masing merupakan vektor eigen dan nilai *eigen* dari matriks kovarian data pelatihan (Gonzales at all, 2002). Matriks kovarian dicari dengan persamaan (2.3).

$$\mu = \frac{1}{M} \sum_{i=1}^{M} \phi_i \phi_i^T = AA^T \qquad (2.3)$$

Dimana matriks $A = [\Phi_1, \Phi_2, ... \Phi_M]$

Matriks kovarian data © akan berukuran N x N, dan digunakan untuk mencari N vektor *eigen* dan nilai *eigen*. Nilai ini memiliki dimensi yang sangat besar untuk proses citra. Untuk itu secara komputasi, diperlukan metode yang lebih sederhana untuk mendapatkan vektor *eigen*-vektor *eigen* ini. Untuk mengatasi hal ini, dapat digunakan dimensi matriks yang lebih kecil yaitu M x M

untuk proses PCA, dimana M merupakan jumlah citra yang digunakan dalam pelatihan. Oleh Yambor (2000), cara tersebut dikenal dengan metode *Snapshot Eigenface*. Dengan analisis ini, perhitungan menjadi berkurang secara dramatis, dari ukuran jumlah piksel N menjadi M (M<<N). Pemecahan metode ini digunakan jika jumlah data pelatihan jauh lebih kecil dari dimensi vektor wajah.

Citra wajah didefinisikan berada dalam suatu ruang citra (*image space*). Metode *Eigenface* mereduksi dimensi citra dengan mentransformasikan citra dari ruang citra berdimensi n (*n-dimensional image space*) ke ruang ciri berdimensi m (*m-dimensional feature space*), dimana nilai m < n [3]. Hasil transformasi disebut sebagai vektor ciri.

2.3.1.1 Reduksi Dimensi dengan Metode Eigenface

Misalkan *learning set* terdiri dari N citra wajah : $\{x_1, x_2, \ldots, x_N\}$, masing-masing terdiri dari n pixel, dimana n = lebar*tinggi citra. Asumsi bahwa setiap citra merupakan anggota salah satu dari C kelas citra wajah : $\{X_1, X_2, \ldots, X_c\}$.

Setiap citra dapat direpresentasikan sebagai vektor baris x_i, i = 1...N, berdimensi n. Nilai n merupakan dimensi ruang citra, sehingga x_i berada dalam ruang citra berdimensi n.

Rata-rata vektor citra, μ, dapat diperoleh melalui persamaan berikut :

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad (2.4)$$

Selisih vektor citra dengan rata-rata vektor adalah :

$$\Phi_i = x_i - \mu \qquad (2.5)$$

dimana i = 1...N.

Matriks kovarian S_T, disebut juga sebagai matriks *total-scatter* citra, didefinisikan sebagai :

$$S_T = A^T A \qquad (2.6)$$

dimana $A = [\Phi_1, \Phi_2, \ldots, \Phi_N]$.

Transformasi *Karhunen-Loeve* atau PCA terhadap vektor citra akan menghasilkan vektor-vektor ciri yang memiliki *total-scatter* **:**

$$WS_T W^T$$

dengan *W* adalah matriks tranformasi.

Matriks transformasi *W* yang dipilih adalah matriks dengan kolom-kolom ortonormal yang dapat memaksimalkan determinan dari *total-scatter* vektor-vektor ciri [3]. Atau :

$$W = \arg\max_W |WS_T W^T|$$

$$= [w_1; w_2; \ldots; w_m] \qquad (2.7)$$

dimana w_i, i = 1…m, adalah kumpulan vektor eigen dari S_T (dalam bentuk vektor baris) yang bersesuaian dengan m nilai eigen terbesar. Vektor-vektor *eigen* ini, yang disebut juga *pricipal components*, memiliki dimensi yang sama dengan dimensi citra wajah, yaitu n, sehingga disebut sebagai *eigenfaces* atau *eigenpictures*.

Eigenfaces merupakan vektor-vektor basis dari ruang ciri dimensi-m [9]. Transformasi citra dari ruang citra dimensi-n ke ruang ciri dimensi-m adalah :

$$y_i = \Phi_i W^T \qquad (2.8)$$

dimana i = 1...N.

Dengan demikian, dapat diperoleh vektor ciri berdimensi m untuk masing-masing citra.

Besarnya nilai m dapat ditentukan melalui persamaan berikut :

$$m = \min_r \left\{ \frac{\sum_{i=1}^{r} d_i}{\sum_{i=1}^{N} d_i} > \theta \right\} \qquad (2.9)$$

dimana θ adalah suatu nilai ambang atau *threshold* dan 0 < *threshold* ≤ 1 [9].

2.3.1.2 Metode *Eigenface* untuk Citra dengan Variasi Pencahayaan dan Ekspresi Wajah

Matriks transformasi *W* atau eigenfaces yang dihasilkan metode Eigenface bertujuan untuk memaksimalkan *total-scatter* atau jarak antar vektor ciri. Sayangnya, transformasi tidak hanya memaksimalkan jarak antar kelas (*between-class scatter*) yang baik untuk klasifikasi, tetapi juga memaksimalkan jarak intra kelas (*within-class scatter*) yang kurang baik untuk klasifikasi [3].

Jika citra masukan merupakan citra-citra dengan variasi pencahayaan atau ekspresi wajah, vektor ciri yang dihasilkan akan dibedakan berdasarkan pencahayaan atau ekspresi. Apabila perubahan pencahayaan atau ekspresi wajah cukup ekstrim bila dibandingkan dengan citra dalam kondisi normal,

pengelempokkan vektor ciri pada ruang ciri kurang baik, bahkan bisa bercampur antara satu kelas citra dengan kelas citra lain.

Sebuah solusi telah ditawarkan, yaitu dengan menghilangkan 3 *principal component (eigenvector)* pertama dari matriks S_T, dimana 3 buah eigenvektor ini bersesuaian dengan 3 nilai eigen terbesar. Jika 3 *principal component* tersebut menyimpan informasi variasi pencahayaan atau ekspresi, pengelompokkan vektor ciri yang lebih baik dapat diperoleh dengan menghilangkan ketiga *principal component* tersebut. Tetapi hal ini akan mengakibatkan hilangnya informasi lain, yang tidak berkaitan dengan variasi pencahayaan atau ekspresi, yang penting untuk pengelompokan (*clustering*).

2.3.1.3 Algoritma *Eigenface*

Dasar dari metode *Eigenface* adalah *Principal Component Analysis* (PCA) yang disebut juga transformasi *Karhunen-Loeve*. Metode ini bertujuan untuk mentrasformasi vektor citra dari ruang citra dimensi-n ke ruang ciri dimensi-m. Algoritma metode *Eigenface* adalah sebagai berikut :

Mencari matriks transformasi (eigenfaces)

Langkah 1: Cari rata-rata vektor citra

$$\mu_{PCA} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad (1*n) \qquad (2.10)$$

Langkah 2 : Untuk i = 1…N, cari selisih vektor citra dengan rata-rata vektor citra, μ

$$\Phi_i = x_i - \mu \qquad (1*n) \qquad (2.11)$$

Simpan Φ_i dalam matriks *A*, sehingga

$$A = [\Phi_1; \Phi_2; \ldots; \Phi_N] \text{ (N*n)} \quad (2.12)$$

Langkah 3 : Cari matriks *total scatter*, S_T

$$S_T = A^T A \quad \text{(n*N)*(N*n) = (n*n)} \quad (2.13)$$

Langkah 4 : Cari vektor eigen dan nilai eigen dari matriks S_T, kemudian urutkan berdasarkan nilai eigen (dari yang terbesar)

$$[v, d] = eig(S_T) \quad (2.14)$$

Nilai eigen dari S_T :

$$d = \begin{bmatrix} d_1 & d_2 & \ldots & d_n \end{bmatrix} \text{ (1*n)} \quad (2.15)$$

dimana $d_1 > d_2 > \ldots > d_n$.

Vektor eigen yang bersesuaian dengan nilai eigen (dalam bentuk vektor kolom) :

$$v = \begin{bmatrix} w_{11} & w_{12} & \ldots & w_{1n} \\ w_{21} & w_{22} & \ldots & w_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ w_{n1} & w_{n2} & \ldots & w_{nn} \end{bmatrix} (n*n) \quad (2.16)$$

Langkah 5 : Tentukan dimensi ruang ciri (vektor eigen yang efektif pengenalan: maksimal N)

$$m = \min_{r} \left\{ \frac{\sum_{i=1}^{r} d_i}{\sum_{i=1}^{N} d_i} > \theta \right\} \qquad (2.17)$$

Langkah 6 : Reduksi vektor eigen dari S_T sesuai dengan nilai m

$$v' = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{bmatrix} (n * m) \qquad (2.18)$$

Matriks transformasi *W* atau eigenfaces adalah :

$$W = v'^{T} \qquad \text{(m*n)} \qquad (2.19)$$

Transformasi

Langkah 7 : Transformasi vektor citra menjadi vektor ciri

$$reduksi = AW^{T} \qquad \text{(N*n)*(n*m) = (N*m)} \qquad (2.20)$$

Atau untuk i = 1…N :

$$y_i = \Phi_i W^{T} \qquad \text{(1*n)*(n*m) = (1*m)} \qquad (2.21)$$

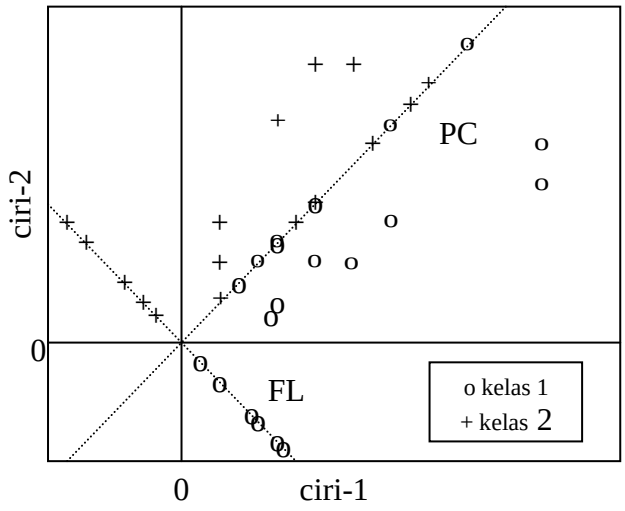
2.3.2 Metode *Fisherface*

Dasar dari metode *Fisherface* ini adalah *Fisher's Linear Discriminant* (FLD). Metode ini ditemukan oleh *Robert Fisher* pada tahun 1936 untuk klasifikasi taksonomi dan menjadi salah satu tehnik yang banyak digunakan dalam pengenalan pola (*pattern recognition*) [3].

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, dapat diekspresikan pada Gambar 2.4.



Gambar 2.4 Perbandingan Metode PCA dan FLD untuk Two-Class Problem

Dari gambar di atas terlihat bahwa transformasi dengan metode PCA menyebabkan kedua kelas saling bercampur. Sedang dengan metode FLD, kedua kelas terpisah (dikelompokkan) dengan baik. Hal ini menunjukkan bahwa walaupun transformasi PCA menghasilkan *total-scatter* yang lebih besar,

transformasi dengan FLD dapat menghasilkan *between-class scatter* yang lebih besar sehingga proses klasifikasi menjadi lebih sederhana [3].

2.3.2.1 Reduksi Dimensi dengan *Fisher's Linear Discriminant*

Fisher's Linear Discriminant, yang menjadi dasar dari metode *Fisherface*, memilih matriks transformasi *W* yang dapat memaksimalkan rasio antara determinan *between-class scatter* dengan *within-class scatter* dari vektor-vektor ciri. Pernyataan tersebut dapat diekspresikan sebagai berikut :

$$W_{opt} = \arg\max_{W} \frac{\left|WS_B W^T\right|}{\left|WS_W W^T\right|} \qquad (2.22)$$

$$= [w_1; w_2; \ldots; w_m]$$

dimana w_i, i = 1...m, merupakan m buah vektor eigen dari rasio antara S_B dengan S_W (dalam bentuk vektor baris), yang bersesuaian dengan m buah nilai eigen terbesar.

Jika w_i adalah vektor eigen dari rasio antara matriks S_B dengan matriks S_W dan d_i merupakan nilai eigen yang bersesuaian, maka :

$$S_B w_i^T = d_i S_W w_i^T$$

dimana i = 1...m dan $d_1 > d_2 > \ldots > d_m$.

Jika x_i, i = 1...N, adalah vektor citra dimensi-n dan masing-masing vektor citra merupakan anggota salah satu dari C kelas citra wajah $\{X_1, X_2, \ldots, X_C\}$, vektor μ adalah rata-rata vektor citra yang dapat diperoleh dari persamaan (2.10), maka matriks *between-class scatter*, S_B, dan matriks *within-class scatter*, S_W, dapat diperoleh melalui persamaan berikut :

$$S_B = \sum_{i=1}^{C} N_i (\mu_i - \mu)^T (\mu_i - \mu) \quad (2.23)$$

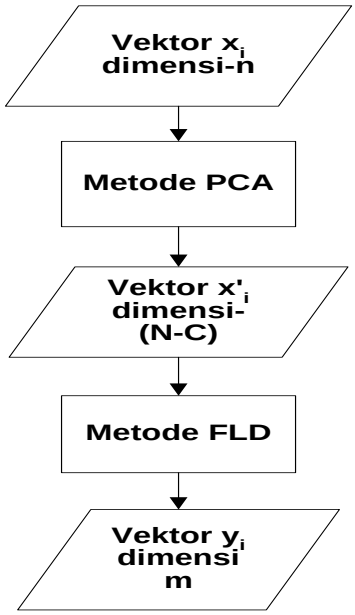
$$S_W = \sum_{i=1}^{C} \sum_{j=1, in_j \in X_i}^{N_i} (x_j - \mu_i)^T (x_j - \mu_i) \quad (2.24)$$

dimana N_i adalah jumlah anggota kelas X_i dan μ_i adalah rata-rata citra anggota kelas X_i, i = 1…C.

Pada pengenalan wajah, matriks *within-class scatter*, S_W, yang dihasilkan selalu singular. Hal ini disebabkan oleh rank maksimum dari matriks S_W adalah (N-C) dan pada umumnya, jumlah citra masukan, N, jauh lebih kecil dari dimensi citra, n [3].

Untuk mengatasi hal ini, vektor-vektor citra terlebih dahulu ditransformasikan ke ruang ciri berdimensi -(N-C) dengan metode PCA, kemudian menerapkan persamaan (2.9) dengan vektor input baru, yaitu x'_i, i = 1..N, berdimensi (N-C). Inilah yang disebut sebagai metode *Fisherface*.

Secara garis besar, proses metode *Fisherface* dapat dinyatakan dalam bagan pada Gambar 2.5 berikut :



Gambar 2.5 Proses Reduksi Dimensi Metode *Fisherface*

Jika W_{PCA} adalah matriks transformasi hasil metode PCA dan matriks transformasi metode FLD, *Wopt*, pada persamaan (2.22) disebut sebagai W_{FLD}, maka pernyataan matematis dari bagan di atas adalah :

$$x'_i = (x_i - \mu) W_{PCA}{}^T \qquad (2.25)$$

Transformasi vektor citra dari ruang citra dimensi-n ke ruang ciri dimensi-(N-C). Dan :

$$y_i = (x'_i) W_{FLD}{}^T \qquad (2.26)$$

Transformasi vektor input baru dari ruang ciri dimensi-(N-C) ke ruang ciri dimensi-m.

Dimensi ruang ciri metode *Fisherface*, m, mempunyai nilai maksimum C-1, karena hanya C-1 nilai *eigen* dari rasio matriks S_B dengan matriks S_W, yang bernilai positif [3].

2.3.2.2 Algoritma *Fisherface*

Dasar dari metode Fisherface adalah *Linear Discriminat Analysis*. Dalam metode *Fisherface* terdiri dari 2 tahap, yaitu :

a. Metode PCA untuk mentransformasi vektor citra dari ruang citra dimensi-n ke ruang ciri dimensi-(N-C)
b. Metode *Fisher's Linear Discriminant* (FLD) untuk mentransformasi vektor masukan baru (hasil transformasi PCA) dimensi-(N-C) ke ruang ciri dimensi-m.

Algoritma metode *Fisherface* adalah sebagai berikut :

Metode PCA

Langkah 1: Cari rata-rata vektor citra

$$\mu_{PCA} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad (1*n) \qquad \text{(dari persamaan 2.10)}$$

Langkah 2 : Untuk i = 1…N, cari selisih vektor citra dengan rata-rata vektor citra, μ_{PCA}

$$\Phi_i = x_i - \mu_{PCA} \qquad (1*n) \qquad \text{(dari persamaan 2.11)}$$

Simpan Φ dalam matriks *A*, sehingga

$$A = [\Phi_1; \Phi_2; \ldots; \Phi_N] \quad (N*n) \qquad \text{(dari persamaan 2.12)}$$

Langkah 3 : Cari matriks *total scatter*, S_T

$$S_T = A^T A \qquad (n*N)*(N*n) = (n*n) \quad \text{(dari persamaan 2.13)}$$

Langkah 4 : Cari vektor *eigen* dan nilai *eigen* dari matriks S_T, kemudian urutkan berdasarkan nilai eigen, dari yang terbesar

$$[v, d] = eig(S_T)$$ (dari persamaan 2.14)

Nilai eigen dari S_T :

$$d = [d_1 \quad d_2 \quad ... \quad d_n] \text{ (1*n)}$$ (dari persamaan 2.15)

dimana $d_1 > d_2 > ... > d_n$.

Vektor eigen yang bersesuaian dengan nilai *eigen* (dalam bentuk vektor kolom) :

$$v = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1n} \\ w_{21} & w_{22} & ... & w_{2n} \\ ... & ... & ... & ... \\ w_{n1} & w_{n2} & ... & w_{nn} \end{bmatrix} (n * n)$$ (dari persamaan 2.16)

Langkah 5 : Reduksi vektor *eigen* dari S_T sesuai dengan nilai (N-C)

$$v' = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1(N-C)} \\ w_{21} & w_{22} & ... & w_{2(N-C)} \\ ... & ... & ... & ... \\ w_{n1} & w_{n2} & ... & w_{n(N-C)} \end{bmatrix} \text{(n*(N-C))}$$ (dari persamaan 2.18)

Matriks transformasi W_{PCA} adalah :

$$W_{PCA} = v'^T \qquad \text{((N-C)*n)}$$ (dari persamaan 2.19)

<u>Transformasi PCA</u>

Langkah 6 : Transformasi vektor citra dimensi-n menjadi vektor masukan dimensi-(N-C), untuk masukan masukan metode FLD

$$input_{FLD} = AW_{PCA}{}^{T} \quad \text{(N*n)*(n*(N-C)) = (N*(N-C))} \qquad (2.27)$$

Atau untuk i = 1…N :

$$x'_i = \Phi_i W_{PCA}{}^{T} \quad \text{(1*n)*(n*(N-C)) = (1*(N-C))} \qquad (2.28)$$

Jadi :

$$input_{FLD} = \begin{bmatrix} x'_1 \\ x'_2 \\ \dots \\ x'_N \end{bmatrix} \qquad (2.29)$$

Metode FLD

Langkah 7 : Cari rata-rata $input_{FLD}$

$$\mu_{FLD} = \frac{1}{N}\sum_{i=1}^{N} x'_i \qquad \text{(1*(N-C))} \qquad (2.30)$$

Langkah 8 : Untuk i = 1..C, cari rata-rata kelas X_i

$$\mu_i = \frac{1}{N_i}\sum_{j=1,x'_j \in X_i}^{N_i} x'_j \qquad \text{(1*(N-C))} \qquad (2.31)$$

dimana N_i adalah jumlah anggota kelas X_i.

Simpan μ_i, i = 1…N, dalam matriks μ_K, sehingga :

$$\mu_K = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_c \end{bmatrix} \quad \text{(C*(N-C))} \qquad (2.32)$$

Langkah 9 : Cari matriks *between-class scatter*, S_B

$$S_B = \sum_{i=1}^{C} N_i \left(\mu_i - \mu_{FLD}\right)^T \left(\mu_i - \mu_{FLD}\right)$$

((N-C)*C)* (C*(N-C)) = ((N-C)*(N-C)) (2.33)

Langkah 10 : Cari matriks *within-class scatter*, S_W

$$S_W = \sum_{i=1}^{C} \sum_{j=1, x'_j \in X_i}^{N_i} \left(x'_j - \mu_i\right)^T \left(x'_j - \mu_i\right)$$

((N-C)*N_i)* (N_i*(N-C)) = ((N-C)*(N-C)) (2.34)

Langkah 11 : Cari rasio antara S_B dengan S_W

$$rasio = \frac{S_B}{S_W} \quad ((N-C)*(N-C)) \qquad (2.35)$$

Langkah 12 : Cari vektor eigen dan nilai eigen dari matriks *rasio*, kemudian urutkan berdasarkan nilai eigen (dari yang terbesar)

$$[v, d] = eig\left(rasio\right) \qquad (2.36)$$

Nilai eigen dari *rasio* :

$$d = \begin{bmatrix} d_1 & d_2 & \ldots & d_{(N-C)} \end{bmatrix} \quad (1*(N-C)) \qquad (2.37)$$

dimana $d_1 > d_2 > \ldots > d_{(N-C)}$.

Vektor eigen yang bersesuaian dengan nilai *eigen* (dalam bentuk vektor kolom) :

$$v = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1(N-C)} \\ w_{21} & w_{22} & \cdots & w_{2(N-C)} \\ \cdots & \cdots & \cdots & \cdots \\ w_{(N-C)1} & w_{(N-C)2} & \cdots & w_{(N-C)(N-C)} \end{bmatrix} ((\text{N-C})*(\text{N-C})) \qquad (2.38)$$

Langkah 13 : Tentukan dimensi ruang ciri (vektor *eigen* yang efektif untuk pengenalan : maksimal C)

$$m = \min_r \left\{ \frac{\sum_{i=1}^{r} d_i}{\sum_{i=1}^{C} d_i} > \theta \right\} \qquad (2.39)$$

Langkah 14 : Reduksi vektor eigen dari *rasio* sesuai dengan nilai m

$$v' = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ w_{(N-C)1} & w_{(N-C)2} & \cdots & w_{(N-C)m} \end{bmatrix} ((\text{N-C})*\text{m}) \qquad (2.40)$$

Matriks transformasi W_{FLD} adalah :

$$W_{FLD} = v'^{T} \qquad (\text{m}*(\text{N-C})) \qquad (2.41)$$

<u>Transformasi FLD</u>

Langkah 15 : Transformasi vektor masukan dimensi-(N-C) menjadi vektor ciri dimensi-m

$$reduksi = input_{FLD} W_{FLD}^{\ T}$$

$$(\text{N}*(\text{N-C}))*((\text{N-C})*\text{m}) = (\text{N}*\text{m}) \qquad (2.42)$$

Atau untuk i = 1…N :

$$y_i = x'_i W_{FLD}^T$$

(1*n)*(n*(N-C)) = (1*(N-C)) (2.43)

2.4 *Image Processing*

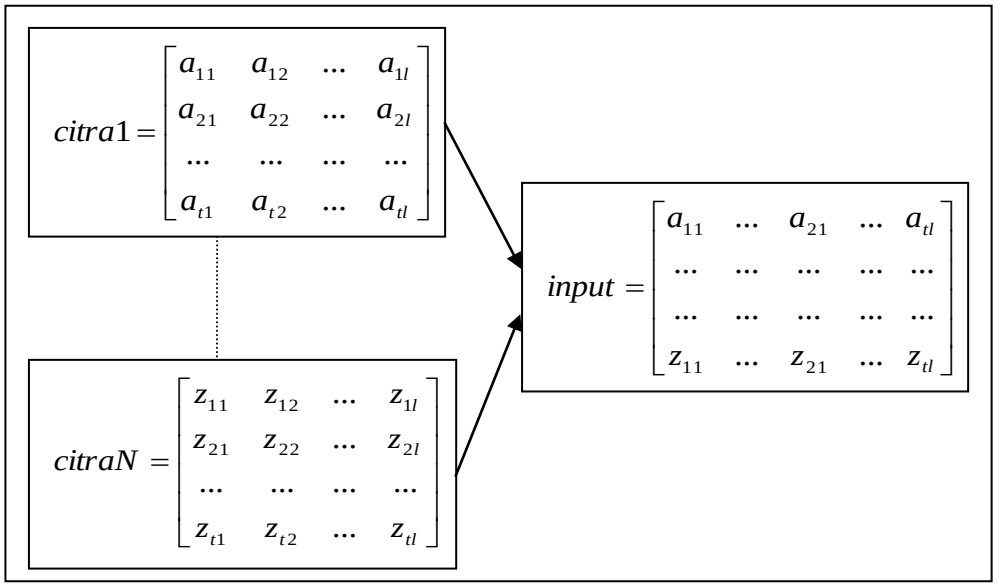
Image *gresccale* lebih mudah untuk dianalisa apabila dibandingkan dengan *image* berwarna. *Image* berwarna adalah kombinasi tiga warna utama merah, hijau dan biru yang biasa disebut dengan sistem warna RGB. Terdiri dari 24 bit, masing-masing 8 bit untuk merah, hijau dan biru. Akan lebih mudah untuk dianalisa maka *image* terlebih dahulu diubah menjadi *image greyscale* dengan cara memberi nilai yang sama untuk masing-masing nilai 8 bit tersebut.

2.4.1 Pemetaan Citra menjadi Vektor Ciri

Dalam proses pengenalan pola wajah menerima masukan berupa citra bitmap (hasil *capture*) yang diubah dalam format *grey-level*. Kumpulan citra yang menjadi masukan pada proses pencarian matriks transformasi disebut *training set*, dan N = jumlah citra *training set*. Kumpulan citra yang menjadi masukan uji pengenalan disebut *testing set*, dan T = jumlah citra *testing set*.

Citra masukan dipetakan kedalam nilai-nilai bertipe *float*, yang merepresentasikan tingkat ke-abuan dari *pixel* pada posisi yang bersesuaian. Nilai-nilai ke-abuan dari suatu citra kemudian disimpan dalam bentuk vektor baris, yang disebut vektor ciri.

Misalkan terdapat N buah citra wajah yang akan dijadikan masukan sistem. Masing-masing citra memiliki dimensi n. Gambar dibawah ini menggambarkan penyimpanan vektor-vektor citra kedalam matriks *input* berdimensi N*n.



Gambar 2.6 Proses Pemetaan Citra menjadi Vektor Ciri

2.4.2 Algoritma Transformasi *Testing Set*

Misalkan *testing set* terdiri dari T citra wajah manusia : $\{x_1, x_2, \ldots, x_T\}$ dengan dimensi n. Jika telah diperoleh :

a. Dari metode *Eigenface :*

   Matriks *W* : matriks eigenfaces dengan dimensi me*n

b. Dari metode *Fisherface :*

   Matriks W_{PCA} : matriks transformasi PCA dengan dimensi (N-C)*n

   Matriks W_{FLD} : matriks transformasi FLD dengan dimensi mf*(N-C)

   dimana N = jumlah citra dan C = jumlah kelas pada *learning set*.

Algoritma transformasi *testing set* dari ruang citra dimensi-n ke ruang ciri dimensi-me atau ruang ciri dimensi-mf adalah sebagai berikut :

a. Transformasi ke ruang ciri dimensi-me *(Eigenface)* :

Langkah 1 : Traensformasi vektor citra dimensi-n menjadi vektor ciri dimensi-me

$$reduksi = AW^T \qquad \text{(T*n)*(n*me) = (T*me)}$$

dengan

$$A = [\Phi_1; \Phi_2; \ldots; \Phi_T] \qquad \text{(T*n)}$$

Atau untuk i = 1…T :

$$y_{Ti} = \Phi_i W^T \qquad \text{(1*n)*(n*me) = (1*me)}$$

Vektor Φ_i adalah :

$$\Phi_i = x_i - \mu \qquad \text{(1*n)}$$

dan μ adalah rata-rata citra *learning set*.

b. Transformasi ke ruang ciri dimensi-mf *(Fisherface)* :

Langkah 1 : Transformasi vektor citra dimensi-n menjadi vektor ciri dimensi-(N-C), untuk masukan FLD

$$input_{FLD} = AW_{PCA}{}^T$$

(T*n)*(n*(N-C)) = (T*(N-C))

dengan $A = [\Phi_1; \Phi_2; \ldots; \Phi_T]$ (T*n)

Atau untuk i = 1…T :

$$x'_i = \Phi_i W_{PCA}{}^T$$

(1*n)*(n*(N-C)) = (1*(N-C))

Vektor Φ_i adalah :

$$\Phi_i = x_i - \mu \quad (1*n)$$

dan μ adalah rata-rata citra *learning set*.

Langkah 2 : Transformasi vektor $input_{FLD}$ dimensi-(N-C) menjadi vektor ciri dimensi-mf

$$reduksi = inputFLDW_{FLD}^{T}$$

(T*(N-C))*((N-C)*m) = (T*m)

Atau untuk i = 1…T :

$$y_{T\,i} = x'_i\, W_{FLD}^{T}$$

(1*n)*(n*m) = (1*m)

2.4.3 Pengenalan Wajah *(Recognition)*

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.

Berdasarkan jumlah *feature* yang diambil, presentase pengenalan dari PCA dapat berubah-ubah. Jumlah *feature* yang optimal akan didapatkan dari hasil eksperimen beberapa kali.

$$Distance = |f_x - f_{database}|$$

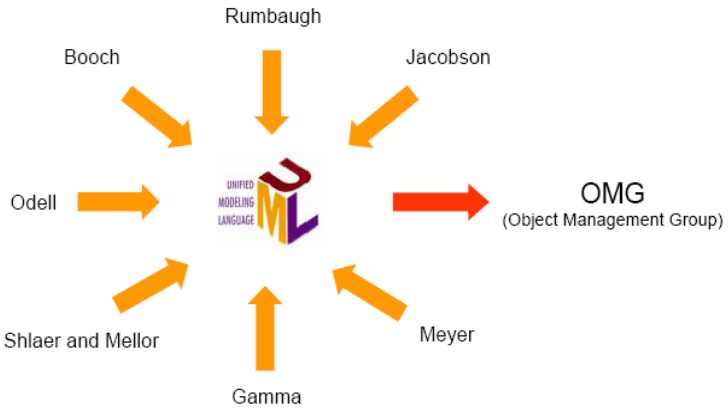
$$= \sqrt{(f_{x1} - f_{database1})^2 + f_{x1} - f_{database2})^2 + ... + (f_{x1} - f_{databasez})^2}$$

Dimana *z* adalah jumlah *feature* yang diambil

Setelah didapat semua *distance* maka dicari yang paling minimum, maka dilihat *distance* minimum tersebut bersesuaian dengan image ke-berapa dan diambil nama orang dari *image* tersebut.

2.5 *Unified Modeling Language* (UML)

UML adalah bahasa grafis untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha bersama dari banyak pihak. Standar UML dikelola oleh OMG (*Object Management Group*).[5]



Gambar 2.7 *Object Management Group*

UML adalah bahasa pemodelan untuk menspesifikasikan, memvisualisasikan, membangun, dan mendokumentasikan artifak-artifak dari sistem.[5]

1. Di dalam *system intensive process*, metode diterapkan sebagai proses untuk menurunkan atau mengevolusikan sistem.
2. Sebagai bahasa, UML digunakan untuk komunikasi yaitu alat untuk menangkap pengetahuan mengenai satu subjek dan mengekspresikan pengetahuan (sintaks) yang memperdulikan subjek untuk maksud komunikasi. Subjek adalah sistem yang dibahas.
3. Sebagai bahasa pemodelan, UML fokus pada pemahaman subjek melalui formulasi model dari subjek. Model memuat pengetahuan pada subjek, dan aplikasi dari pengetahuan ini berkaitan dengan *intelegency*.
4. Berkaitan dengan unifikasi, UML memadukan praktek rekayasa terbaik sistem informasi dan industri, meliputi beragam tipe sistem (perangkat lunak dan non perangkat lunak), domain (bisnis, perangkat lunak) dan proses siklus hidup.
5. Ketika diterapkan untuk menspesifikasikan sistem, UML dapat digunakan untuk mengkomunikasikan "apa" yang diperlukan dari sistem dan "bagaimana" sistem dapat direalisasikan.
6. Ketika diterapkan untuk memvisualisasikan sistem, UML dapat digunakan untuk menjelaskan sistem secara visual sebelum direalisasikan.
7. Ketika diterapkan untuk membangun sistem, UML dapat digunakan untuk memandu realisasi sistem serupa dengan "*blueprint*".

8. Ketika diterapkan untuk mendokumentasikan sistem, UML dapat digunakan untuk menangkap pengetahuan mengenai sistem pada seluruh siklus hidup.

2.5.1 Tujuan UML

Tujuan utama perancangan UML adalah :

1. Menyediakan bahasa pemodelan visual yang ekspresif dan siap pakai untuk mengembangkan dan pertukaran model-model yang berarti.
2. Menyediakan mekanisme perluasan dan spesifikasi untuk memperluas konsep-konsep inti.
3. Mendukung spesifikasi independen bahasa pemrograman dan proses pengembangan tertentu.
4. Menyediakan basis formal untuk pemahaman bahasa pemodelan.
5. Mendorong pertumbuhan pasar kakas berorientasi objek.
6. Mendukung konsep-konesp pengembanga level lebih tinggi seperti komponen, kolaborasi, *framework* dan *pattern*.

2.5.2 Diagram Perilaku

Diagram ini memvisualisasikan, menspesifikasi, membangun dan mendokumentasikan aspek dinamis dari sistem.

1. *Class diagram*

Class adalah sebuah spesifikasi yang jika diinisialisasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi

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

Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi dan lain-lain.

2. *Statechart Diagram*

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram)*.

3. *Activity Diagram*

Activity diagram menggambarkan berbagai aliran alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses pararel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram khusus*, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya *(internal processing)*. Oleh karena itu *activity diagram* tidak menggambarkan behavior internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *usecase* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *usecase* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

4. *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atas dimensi vertical (waktu) dan dimensi horizontal (objek-objek terkait).

Sequence diagram bias digunakan untuk menggambarkan scenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yan men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

5. *Collaboration Diagram*

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number*, dimana *message* dari level tertinggi memiliki nomor 1, *message* dari level yang sama memiliki prefiks yang sama.

6. *Component Diagram*

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan *(dependency)* di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time, link time*, maupun *run-time*.

Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tetapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

7. *Deployment Diagram*

Deployment diagram menggambarkan detail bagaimana komponen di-*delay* dalam infrastruktur sistem, dimana komponen akan terletak (pada mesin, server atau piranti keras), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server dan hal-hal lain yang bersifat fisikal.