BAB II

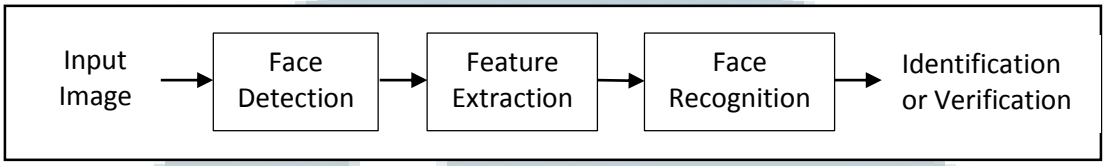
LANDASAN TEORI

2.1 Face Recognition

Menurut Pandya dkk. (2013) *Face recognition* merupakan teknologi identifikasi yang paling relevan dalam bentuk analisis gambar, dimana *face recognition* ini sendiri merupakan suatu sistem otomatis yang dibangun menyerupai kemampuan manusia dalam proses pengenalan wajah. *Face recognition* juga menjadi teknologi biometrik yang paling penting dan baik untuk digunakan sebagai standar dalam realita dan persepsi sosial, serta peningkatan keamanan akan data-data yang bersifat *confidential*.

Dalam hal yang lebih spesifik, Pandya dkk. (2013) memaparkan tentang beberapa unsur dari *face recognition* itu sendiri. Sistem dalam *face recognition* terdiri dari dua kategori, yaitu verifikasi dan identifikasi. Proses verifikasi merupakan pencocokkan 1:1 untuk membandingkan apakah wajah tersebut benar merupakan wajah manusia berdasarkan *template* wajah yang sudah ada. Sebaliknya, proses identifikasi merupakan pencocokkan 1:N untuk membandingkan wajah terdeteksi dengan semua wajah yang tersimpan dalam *database* apakah ada yang cocok atau tidak. Teknik dalam *face recognition* secara umum dapat dibagi menjadi tiga kategori berdasarkan metode akuisisi data wajah, yaitu: metode yang dijalankan berdasarkan intensitas gambar, metode dengan rangkaian video, dan metode yang membutuhkan sensor data lainnya sebagai tambahan, seperti informasi 3D atau perbandingan *infra-red.*

Pandya dkk. (2013) berpendapat bahwa selama 10 tahun terakhir, teknologi *face recognition* menjadi salah satu topik popular dalam penelitian terkait bidang komputer visual. Berikut ini adalah gambaran dasar dari model *face recognition* dari awal hingga akhir proses.



Gambar 2.1 Alur kerja *face recognition*
Sumber : Pandya (2013, p. 1)

Menurut Lu dkk. (2003), ada banyak metode dan algoritma yang dapat digunakan dalam sebuah aplikasi *face recognition*. Untuk mendapatkan hasil yang akurat, perlu dipertimbangkan dua hal penting dalam mengaplikasikan metode-metode tersebut, yaitu: pemilihan fitur yang tepat untuk representasi wajah, dan proses klasifikasi citra wajah berdasarkan representasi fitur yang dipilih.

Dari beberapa solusi terhadap masalah *face recognition*, pendekatan dari tampilan dan kinerja yang paling baik dalam pemrosesan objek wajah atau gambar adalah dengan merepresentasikannya ke dalam pola dua dimensi secara keseluruhan untuk menghindari kesulitan terkait model tiga dimensi dan bentuk atau objek-objek yang tidak digunakan lainnya.

Cevikalp dkk. (2010) berpendapat bahwa secara tradisional, *face recognition* merupakan suatu proses identifikasi wajah dari sebuah gambar tunggal, dan banyak diasumsikan bahwa gambar harus diambil dalam suatu kondisi lingkungan yang terkendali. Bagaimanapun, tampilan citra wajah dapat berubah secara drastis dalam beberapa variasi pose, pencahayaan, ekspresi, dan lain sebagainya.

Gambar yang diambil dengan suatu kondisi lingkungan yang terkontrol tidak dapat mencukupi kebutuhan dalam proses identifikasi wajah karena ada banyak faktor yang mempengaruhi citra wajah seseorang seperti yang sudah dipaparkan sebelumnya, apalagi gambar akan diambil secara *real time* dengan situasi dan kondisi yang sedang terjadi.

Masih menurut Cevikalp dkk. (2010), dalam beberapa waktu belakang ini dikembangkan sistem *face recognition* dengan menggunakan beberapa set citra wajah. Dengan metode yang didasarkan pada kumpulan gambar ini, diharapkan bisa meningkatkan performa dari proses *face recognition* karena banyaknya variasi informasi terhadap berbagai penampilan wajah individu. Dengan begitu, proses pengambilan keputusan dalam identifikasi wajah bisa dilakukan dengan perbandingan terdekat dari kumpulan gambar yang menjadi patokan dalam identifikasi.

2.2 Principal Component Analysis

Menurut Sandhu dkk. (2009), *principal component analysis* atau PCA merupakan teknik yang paling baik dan sering digunakan untuk keperluan identifikasi gambar dan kompresi data. PCA sendiri bisa dikategorikan sebagai sebuah metode statistik yang masih berhubungan dengan analisis faktor. Fungsi utama dari PCA adalah untuk mengurangi ukuran data dari suatu *data space* atau variabel ke dimensi yang lebih kecil, yang nantinya akan digunakan untuk merepresentasikan data ke dalam bentuk *file* dengan ukuran yang efisien.

Ide utama dari penggunaan *principal component analysis* untuk sistem *face recognition* adalah untuk mengekspresikan vektor 1-D besar berisi kumpulan *pixel*

yang dibangun dari gambar wajah 2-D, menjadi susunan komponen yang padat dari fitur-fitur gambar yang ada. Metode ini disebut dengan *projection of eigenspace.* Nilai *eigenspace* dihitung berdasarkan identifikasi dari *eigenfactor* yang didapat dari matriks *covarient* yang berasal dari kumpulan gambar wajah yang sudah direpresentasikan ke dalam bentuk vektor.

Setiap gambar contoh wajah untuk proses identifikasi diubah menjadi bentuk ruang wajah dan komponen-komponen penting untuk disimpan ke dalam memori. Wajah hasil dari *input* juga akan diubah menjadi bentuk ruang wajah, kemudian dilakukan perhitungan untuk menentukan perbandingan jarak wajah dengan setiap gambar yang tersimpan dalam sistem.

Menurut Kim (2001), sebuah gambar wajah 2-D dapat direpresentasikan menjadi vektor 1-D dengan menggabungkan setiap baris (atau kolom) untuk dijadikan sebuah vektor pipih yang panjang. Berdasarkan teori tersebut, berikut ini adalah rumus yang digunakan untuk menggambarkan proses generalisasi dari PCA. Diasumsikan bahwa ada M vektor dengan ukuran N (N = baris gambar x kolom gambar) yang mewakili sekumpulan gambar wajah yang ada. p_j merepresentasikan nilai dari *pixel.*

$$x_i = [p_1 \dots p_N]^T, i = 1, \dots, M \quad \text{..... Rumus 2.1}$$

Kemudian buat *mean* atau rata-rata terpusat dari semua gambar dengan cara mengurangi *mean* gambar dari setiap vektor pada gambar. m mewakili *mean* dari gambar.

$$m = \frac{1}{M}\sum_{i=1}^{M} x_i \quad \text{..... Rumus 2.2}$$

Dan w_1 mendefinisikan rata-rata terpusat dari gambar.

$$w_i = x_i - m \quad \text{..... Rumus 2.3}$$

Tujuan utamanya adalah untuk mencari *set* dari e_i yang memiliki kemungkinan terbesar untuk dapat diproyeksikan ke dalam setiap w_1, dan nantinya ditemukan *set* dari M vektor ortogonal e_i yang dimana kuantitasnya

$$\lambda_i = \frac{1}{M}\sum_{n=1}^{M}(e_i{}^T w_n)^2 \quad \text{..... Rumus 2.4}$$

dimaksimalkan dengan *orthonomality constraint.*

$$e_l{}^T e_k = \delta_{lk} \quad \text{..... Rumus 2.5}$$

Sudah terlihat bahwa e_i dan λ_i didapatkan dari *eigenvectors* dan *eigenvalues* dari matriks kovarian

$$C = WW^T \quad \text{..... Rumus 2.6}$$

dimana W merupakan matriks yang dibentuk dari kolom vektor-vektor w_i dan disusun bersebelahan secara horizontal. Ukuran dari C adalah N x N yang sangat besar. Sebagai contoh, gambar dengan ukuran 64 x 64 dibuat matriks kovarian dengan ukuran 4096 x 4096. Tidak mudah untuk dapat menyelesaikan *eigenvectors* dari C secara langsung.

Teori dasar dalam aljabar linear mengemukakan bahwa vektor e_i dan nilai scalar λ_i bisa didapatkan dengan cara menyelesaikan perhitungan dari *eigenvectors* dan *eigenvalues* dari matrik W^TW dengan ukuran M x M. Diasumsikan bahwa d_i dan μ_i adalah nilai *eigenvectors* dan *eigenvalues* dari W^TW masing-masing.

$$W^TWd_i = \mu_i d_i \quad \text{..... Rumus 2.7}$$

Kemudian kedua sisi dikalikan dengan W

$$W^TW(Wd_i) = \mu_i(Wd_i) \quad \text{..... Rumus 2.8}$$

Nilai *eigenvectors* sesuai dengan nilai *eigenvalues* ≠ 0 dari matriks kovarian yang menghasilkan basis orthonormal dari *subspace*, dimana paling banyak gambar yang direpresentasikan dengan kemungkinan *error* yang kecil. Nilai *eigenvectors* diurutkan mulai dari yang terbesar sampai terkecil untuk disesuaikan dengan *eigenvalues.* Nilai *eigenvectors* yang terhubung dengan *eigenvalues* terbesar menunjukkan bahwa itu merupakan gambar dengan perbedaan kondisi terbanyak.

Maka dari itu, *eigenvalues* terkecil yang terhubung dengan *eigenvectors* yang dapat digunakan untuk menemukan gambar dengan perbedaan yang paling sedikit. Metode ini dapat mengurangi model eksponensial, yang berarti sekitar 90 % dari total perbedaan yang ada sudah terhitung di dalam 5% sampai 10% dimensi gambar.

Sebuah gambar wajah dapat diproyeksikan ke dalam $M'(\ll M)$ dimensi dengan menghitung

$$\Omega = [v_1 v_2 \ldots v_{M'}]^T \qquad \text{..... Rumus 2.9}$$

dimana $v_i = e_i{}^T w_i$. v_i adalah koordinat ke i dari gambar citra wajah dalam ruang yang baru, yang menjadi *principal component*. Kemudian komponen inilah yang nantinya akan digabungkan dengan algoritma *eigenface* guna mendapatkan hasil dari identifikasi wajah.

2.3 Eigenfaces

Menurut Sandhu dkk. (2009), *eigenface* merupakan salah satu metode yang digunakan untuk identifikasi wajah. Konsep dasar dari *eigenface* ini adalah metode untuk mengurangi suatu informasi pada gambar. Jika ditinjau secara mendalam, dalam gambar dengan ukuran kecil pun, ada sangat banyak informasi yang terdapat

di dalamnya. Dari semua kemungkinan informasi yang terdapat dalam suatu gambar, di dalamnya bisa diklasifikasikan bentuk gambar dengan wajah di dalamnya dan bisa direpresentasikan dalam suatu ruang gambar untuk diperhitungkan. Untuk itu, dibutuhkan metode yang dapat digunakan untuk memisahkan antara gambar wajah dengan gambar biasa. Untuk melakukan metode ini, dibutuhkan sebuah gambar wajah dasar yang kemudian dikombinasikan dengan perhitungan linear. Bentuk gambar wajah dasar yang ada, kemudian diubah menjadi suatu bentuk matriks untuk mengurangi kompleksitas dalam perhitungannya. Singkatnya, penggunaan *eigenface* untuk *face recognition* dapat disimpulkan dengan langkah-langkah utama sebagai berikut.

a. Lakukan perhitungan *eigenface.*

b. Proyeksikan sampel gambar wajah menjadi bentuk ruang wajah (matriks) untuk digunakan pada metode klasifikasi yang sudah disesuaikan.

c. Evaluasi hasil proyeksi, kemudian bandingkan citra wajah dengan sampel gambar wajah yang sudah untuk mendapatkan hasil pencocokkan.

Untuk menjalankan metode ini, tentunya pertama-tama diperlukan terlebih dahulu gambar wajah yang akan digunakan. Gambar wajah yang ada harus dikonversikan ke dalam bentuk *grayscale image,* dengan nilai intensitas *pixel* antara 0 sampai 255.

Diasumsikan, ada *K* jumlah gambar yang digunakan. Setiap sampel wajah akan direpresentasikan menjadi x_i, dimana i sebagai penanda urutan gambar (1 <= i <= *K*). Setiap x_i merupakan sebuah vektor kolom. Secara umum, suatu gambar terdiri dari banyak *pixel* dan memiliki koordinat masing-masing (x,y). Yang harus dilakukan dalam implementasi metode ini adalah mengkonversikan bentuk matrik

tersebut menjadi 1-D. Jika bentuk sampel gambar dengan panjang x dan lebar y, maka bentuk setelah dikonversi akan menjadi (x * y) x 1. Hal ini dibutuhkan agar dapat mengubah suatu gambar biasa menjadi bentuk vektor kolom dari *pixel* gambar.

Langkah-langkah ini diperlukan untuk dapat mencocokkan gambar wajah dengan sampel data yang ada. Berikut ini adalah tampilan visual terhadap identifikasi suatu wajah.

Gambar 2.2 Simulasi pencocokkan wajah
Sumber: Sandhu dkk. (2009, p. 3)

Langkah selanjutnya adalah menghitung perbedaan dari setiap wajah U_i, yaitu $U_i = X_i - X$ (dimana X adalah nilai *mean*) dan bentuk dari matriks U, yaitu $U = [U_1 U_2 \dots U_k]$. Tujuannya adalah untuk menghasilkan nilai *eigenface* yang didapat dari hasil perhitungan *eigenvectors* dari matriks kovarian UU^T. Namun, perhitungan ini tidak bisa dilakukan secara langsung karena ukuran dari UU^T adalah (x * y)*(x * y) dan terlalu besar.

Untuk melakukan perhitungan tersebut, tentunya akan memakan waktu yang lama dalam prosesnya. Maka dari itu, digunakan trik dari Aljabar Linear. Nilai *eigenvectors* dari UU^T bisa didapatkan juga dari perhitungan kombinasi linear *eigenvectors* dari matriks $U^T U$. Hal ini sangat berguna dikarenakan matriks $U^T U$

adalah K x K. Dalam aplikasi praktis di dunia nyata K << (x*y). Jadi, nilai *eigenvector* w_j didapatkan dengan proses sebagai berikut.

$$w_j = \frac{\sum_{l=1}^{K} U_l E_{lj}}{\sqrt{\lambda_j}} \quad \text{..... Rumus 2.10}$$

Dimana E_{lj} adalah nilai ke-l dari *eigenvectors* ke-j dari matriks $U^T U$ dan λ_j adalah *eigenvalues* yang sesuai dengan w_j dan E_j. Berikut ini adalah perhitungan untuk mendapatkan nilai *eigenvector* dari $U^T U$. Nilai *eigenvectors* dari $U^T U$ diasumsikan sebagai E_j (1 <= j <= K) and *eigenvalues* diasumsikan sebagai λ_j.

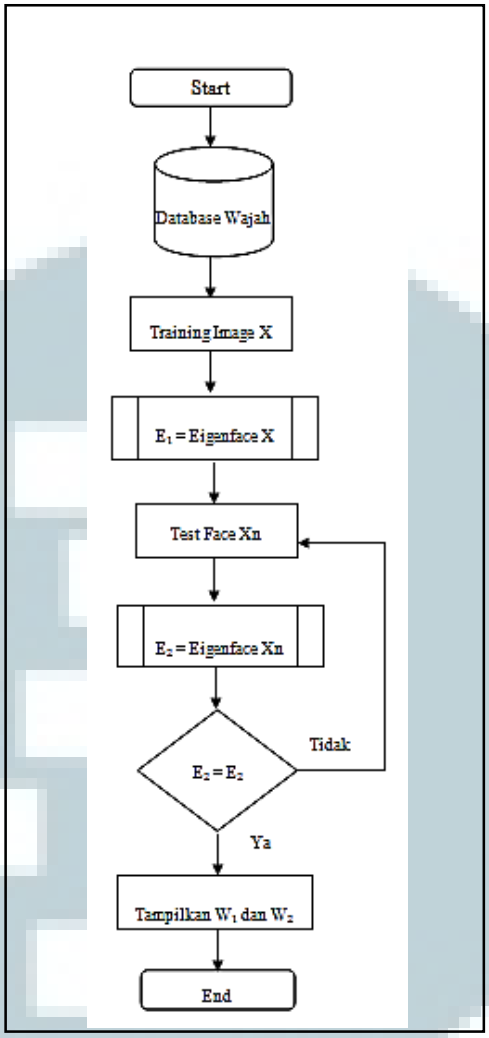
$$U^T U E_j = \lambda_j E_j \quad \text{.... Rumus 2.11}$$

Kemudian kalikan kedua sisi rumus dengan U :

$$U \; x \; U^T U E_j = \lambda_j U E_j \quad \text{.... Rumus 2.12}$$

Jadi, UE_j adalah nilai *eigenvectors* ke-j dari UU^T dengan *eigenvalues* λ_j. Fakta bahwa UU^T dan $U^T U$ mempermudah proses perhitungan dalam mencari *eigenvectors* dan *eigenvalues* dari masing-masing gambar.

Setelah penjelasan secara teknis, berikut ini adalah gambaran alur kerja dari *eigenface*, mulai dari deteksi wajah, implementasi *eigenface*, hingga pencocokkan wajah yang digambarkan melalui sebuah diagram sederhana. Alur prosesnya dapat dilihat sebagai berikut.

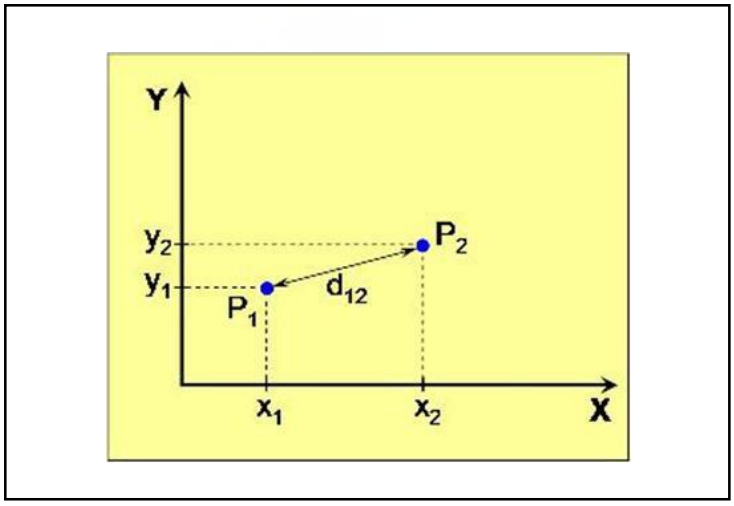


Gambar 2.3 Alur Proses Algoritma *Eigenface*
Sumber : http://www.metode-algoritma.com/2013/02/algoritma-eigenface.html

2.4 Euclidean Distance

Pada perhitungan algoritma *eigenface* yang dilakukan sebelumnya, pencarian jarak yang dimaksud adalah pencarian suatu jarak yang diukur *point to point* dan disebut dengan *Euclidean Distance.* Dalam perhitungan 2 dimensi (2D), jarak *Euclidean* antara titik P1 dan P2 dirumuskan sebagai d_{12} = sqrt(dx^2 + dy^2), dimana dx = x2 – x1, dan dy = y2 – y1. Sedangkan dalam perhitungan 3 dimensi (3D), digunakan rumus sqrt(dx^2 + dy^2 + dz^2), dimana dz = z2 – z1.

Gambar 2.4 berikut ini merepresentasikan gambar dari grafik jarak *Euclidean* 2 dimensi.



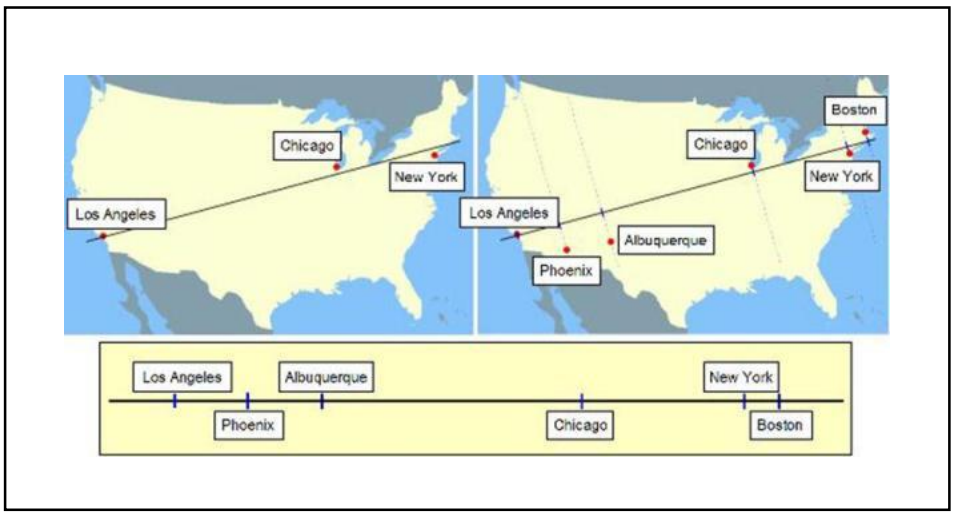
Gambar 2.4 Grafik jarak *Euclidean* 2D
Sumber : http://www.metode-algoritma.com/2013/02/algoritma-eigenface.html

Ketika dilakukan perhitungan jarak dari setiap gambar wajah yang dimiliki, pastinya akan didapatkan banyak perbedaan. Contohnya dalam dua buah gambar terdapat 2000 perbedaan *pixel*. Nilai 2000 ini harus diproses lebih lanjut untuk mendapatkan satu nilai yang nantinya akan digunakan untuk otentikasi. Namun, ada beberapa kendala yang mempengaruhi keakuratan perhitungan ini. Salah satunya adalah *noise*, yang dalam konteks ini sangat berpengaruh adalah *brightness* dari sebuah gambar.

Perbedaan tingkat pencahayaan pada sebuah gambar akan berdampak pada sulitnya proses pencocokkan gambar untuk dilakukan secara akurat. Beberapa *pixel* yang berbeda bisa jadi dideteksi secara sama hanya karena perbedaan *brightness* tersebut. Untuk itu, dalam pengambilan gambar harus juga mempertimbangkan tingkat pencahayaan sekitar agar perbedaan yang terjadi tidak terlalu banyak dan proses perhitungan dalam otentikasi dapat berjalan secara akurat.

2.5 Dimension Reduction

Dengan menggunakan metode *Principal Component Analysis* (PCA), dapat dilakukan pengurangan dimensi sebelum dilakukan proses perhitungan lebih lanjut. Sebagai contoh seperti yang dapat dilihat pada gambar 2.5. Pada gambar 2.5 tersebut, terlihat bahwa kota *Los Angeles*, *Chicago*, dan *New York* dalam peta dapat direpresentasikan sebagai sebuah garis lurus. Untuk mendapatkan semua kota menjadi satu garis lurus secara keseluruhan, maka harus dilakukan pembuangan terhadap faktor-faktor 2D dan 3D yang terdapat dalam peta. Apabila ada kota lain yang akan disertakan dalam garis, maka garis yang sudah ada sebelumnya dapat dijadikan sebagai acuan dalam melakukan normalisasi gambar seperti pada gambar 2.5 sebelah kanan atas. Kemudian hasil representasi dari semua gambar kota dapat dilihat pada gambar 2.5 bagian bawah, dimana semua kota sudah termasuk dalam satu garis lurus, Dengan melakukan hal ini, hasil pemetaan kota-kota tersebut sudah diubah ke dalam bentuk 1D sehingga mempermudah dalam proses perhitungan dan *noise* yang tidak diperlukan dapat dikurangi ataupun dihilangkan bila diperlukan.



Gambar 2.5 Gambar konversi 2D menjadi 1D
Sumber : http://www.metode-algoritma.com/2013/02/algoritma-eigenface.html

Untuk dapat melakukan perubahan dari 2D menjadi 1D, bisa digunakan rumus dengan memanfaatkan garis lurus yang terbentuk yaitu y = mx, dimana m adalah *eigenvector* yang dimiliki.

Proses perubahan dimensi yang terjadi dinamakan proyeksi. Dalam gambar 2.5 tersebut, garis biru merupakan proyeksi kota-kota yang berada dalam daerah garis acuan. Sehingga kota-kota yang tidak berada dalam garis bisa direpresentasikan dalam garis acuan setelah dilakukan proses proyeksi. Itulah mengapa kota *Phoenix*, *Albuquerque*, dan *Boston* dapat ditulis dalam garis lurus dan didapatkan nilai dari garis tersebut.

2.6 Uji Keakuratan Data

Dalam pembuatan suatu aplikasi, tentunya dibutuhkan suatu metode percobaan terhadap implementasi yang dilakukan. Menurut Orso dkk. (2014), metode *testing* dalam pengembangan aplikasi menjadi salah satu fokus guna meningkatkan kualitas dari suatu aplikasi. Untuk menunjang hal tersebut, perlu adanya pengujian terhadap berbagai aspek yang mempengaruhi kinerja dari program. Pengujian ini dapat dilakukan berupa uji performa, uji keakuratan, uji kemudahan dalam penggunaan program, dan uji keamanan. Dalam setiap pengujian ini, tentunya dibutuhkan parameter khusus serta pengalaman pengguna selama menjalankan aplikasi. Maka dari itu, digunakan metode pengambilan data dengan kuisioner. Jumlah kuisioner yang digunakan sebagai sampel dalam pengambilan data juga harus tepat agar uji coba dapat dinilai akurat. Gay dan Diehl (1992) mengemukakan pendapat bahwa sampel yang diambil dalam melakukan uji coba penelitian haruslah sebesar-besarnya. Semakin banyak sampel maka akan semakin representatif dan hasilnya dapat digeneralisir. Masih menurut Gay dan Diehl

(1992), patokan minimum sampel yang diambil agar diterima sangat bergantung pada penelitiannya. Kriterianya dapat dilihat sebagai berikut.

1. Dalam penelitian yang bersifat deskriptif, dibutuhkan sampel minimum sebanyak 10% dari populasi.
2. Jika penelitian bersifat korelasional, dibutuhkan sampel minimum sebanyak 30 subjek.
3. Apabila penelitian bersifat perbandingan, sampel yang dibutuhkan adalah 30 subjek untuk setiap grup.
4. Apabila penelitian bersifat eksperimental, sampel yang dibutuhkan adalah 15 subjek untuk setiap grup.

Fraenkel dan Wallen (1993:92) juga mengemukakan tentang besaran sampel dengan ketentuan minimum sebagai berikut.

1. Penelitian deskriptif sebanyak 100 buah sampel.
2. Penelitian korelasional sebanyak 50 buah sampel.
3. Penelitian kausal-perbandingan sebanyak 30 buah / grup.
4. Penelitian eksperimental sebanyak 15 atau 30 / grup.

Dalam penelitian ini, pendekatan yang dibutuhkan untuk melakukan uji coba adalah bentuk perbandingan dan eksperimental. Perbandingan yang diteliti mencakup perbedaan wajah yang diambil dan diidentifikasi. Sedangkan untuk percobaan adalah tingkat keakuratan pengambilan gambar dan antisipasi peniruan wajah. Berdasarkan fakta para ahli yang sudah dipaparkan sebelumnya, maka dalam pengumpulan data kuisioner, akan digunakan sebanyak 30 data responden, dan 15 data wajah percobaan deteksi untuk setiap responden.

2.7 Peranti Lunak

Menurut Agrawal dkk. (2007) *Software* atau peranti lunak merupakan sekumpulan instruksi untuk menerima *input*, yang kemudian dimanipulasi untuk mendapatkan suatu *output* yang diinginkan dengan fungsi, situasi, kondisi, dan performa yang ditentukan oleh pengguna peranti lunak. Dalam suatu peranti lunak terdapat pula buku petunjuk penggunaan yang bertujuan agar *user* mengerti keseluruhan sistem dari peranti lunak. Peranti lunak sendiri terdiri atas *source code, executeable design document, operation and system manual*, dan *installation and implementation manual*.

Dalam bukunya, Agrawal dkk. (2007) membagi tipe *software* menjadi dua kategori besar, yaitu :

1. Peranti lunak sistem. Peranti lunak ini mencakup sistem operasi dan segala fungsi untuk menjalankan keseluruhan sistem dari komputer.
2. Peranti lunak aplikasi. Peranti lunak ini mencakup program yang bekerja untuk pengguna. Contoh yang paling sederhana adalah *word processing*, dan sistem manajemen basis data yang langsung berhubungan dan memberikan fungsi nyata bagi pengguna.

Agrawal dkk. (2007) juga melakukan klasifikasi terhadap peranti lunak ke dalam dua kelas besar, yaitu:

1. Peranti lunak umum. Peranti lunak ini dirancang untuk pasar yang luas dimana kepentingan dan kebutuhannya sangat umum, stabil, dan dimengerti oleh *software engineering.*
2. Peranti lunak yang disesuaikan. Peranti lunak ini dibuat sesuai dengan kebutuhan pengguna pasar secara khusus dalam ruang lingkup yang kecil.

Berdasarkan defisini peranti lunak di atas, Pressman (2010) menyimpulkan beberapa karakteristik dari peranti lunak yang membedakannya dari peranti keras :

1. Peranti lunak dikembangkan, bukan dimanufaktur secara klasik.

   Walaupun ada beberapa kesamaan dalam pengembangan peranti lunak dan manufaktur peranti keras, secara fundamental kedua aktivitas itu berbeda. Dalam keduanya, kualitas yang tinggi dicapai melalui rancangan yang baik, tapi dalam tahap manufaktur peranti keras, ada banyak masalah kualitas yang muncul dan sulit untuk diperbaiki. Kedua aktivitas ini juga tergantung pada sumber daya manusia dan memerlukan konstruksi produk yang baik. Namun, pendekatan untuk kedua aktivitas tersebut berbeda jauh karena proyek piranti lunak tidak bisa dikerjakan seperti proyek manufaktur.

2. Peranti lunak tidak akan habis walau digunakan terus-menerus.

   Dalam proses manufaktur peranti keras, ada indikasi terjadinya kesalahan yang tinggi di awal pengembangan. Kesalahan ini dapat diperbaiki dan peranti keras dapat hidup pada fase yang aman. Namun seiring dengan berjalannya waktu, tingkat kesalahan dapat kembali meningkat karena efek kumulatif seperti debu, getaran, penyalahgunaan, perubahan temperatur yang ekstrim, dan efek gangguan lingkungan lainnya. Singkatnya, peranti keras lama-lama akan habis jika dipakai terus-menerus.

   Lain halnya dengan peranti lunak. Perubahan lingkungan tidak akan berpengaruh apapun terhadap kinerja dari peranti lunak. Secara teoritis, grafik rata-rata kesalahan pada peranti lunak relatif lebih ideal. Kesalahan-kesalahan yang terdapat pada peranti lunak biasanya dapat terdeteksi sejak awal. Dengan perbaikan-perbaikan secara berkala, dapat membuat kualitas hidup suatu

peranti lunak menjadi lebih baik. Perlu diketahui bahwa suatu peranti lunak tidak selamanya dapat memenuhi kebutuhan pengguna. Selama masa hidupnya tentu akan ada perubahan yang membuat grafik rata-rata kesalahannya meningkat sedikit demi sedikit.

Dengan pemaparan di atas, dapat disimpulkan bahwa peranti lunak tidak akan habis walau dipakai terus menerus, namun kinerjanya dapat menurun akibat perubahan-perubahan yang dilakukan. Kerusakan pada peranti lunak hanya dapat diperbaiki dengan program. Lain halnya dengan peranti keras yang dapat diperbaiki hanya dengan mengganti sebagian perangkatnya dengan yang baru.

3. Walaupun industri bergerak ke arah standarisasi konstruksi, peranti lunak tetap pada prinsip kostumisasi terhadap penggunanya.

Seiring dengan berkembangnya evolusi terhadap ilmu *engineering*, banyak standar desain yang diciptakan. Para ilmuwan bekerja dengan ide dasar untuk menciptakan suatu komponen dasar yang dapat digunakan secara umum untuk berbagai peranti keras. Hal ini menginspirasi para pengembang peranti lunak untuk membuat standar juga. Akan tetapi, walaupun peranti lunak memiliki standar, suatu saat komponen dasar dari peranti lunak tersebut juga dapat dikostumisasi sesuai dengan kebutuhan pengguna.

Kevin Arian (2013) dengan mengutip Sommerville (2007) mengatakan bahwa rekayasa peranti lunak (*software engineering*) adalah disiplin ilmu insinyur yang fokus pada semua aspek dari produksi peranti lunak dari tahap awal, yaitu spesifikasi sistem sampai pada tahap pemeliharaan saat sistem sudah digunakan. Diungkapkan pula bahwa perlu diperhatikan bahwa *computer science* berbeda

dengan *software engineering*. *Computer science* berfokus pada teori dan metode yang mendasari komputer dan sistem peranti lunak, sedangkan *software engineering* berfokus pada masalah praktis dalam produksi peranti lunak. Oleh karena itu, ada dua kata kunci yang perlu diperhatikan dalam terminologi rekayasa peranti lunak.

Pertama adalah disiplin ilmu insinyur yang membuat sesuatu bekerja dengan mengaplikasikan teori, metode, dan berbagai peralatan yang ada. Walaupun demikian, seorang insinyur tidak berhenti pada teori atau metode yang ada. Insinyur terus mencari solusi untuk suatu permasalahan yang belum ada teori untuk memecahkannya. Kedua adalah aspek-aspek dari produksi peranti lunak.

Menurut Sundar (2010, pp. 9-10), ketika suatu spesifikasi peranti lunak sudah didapatkan, pengembangan peranti lunak akan melalui beberapa fase yang secara umum adalah tahap spesifikasi, desain, dan implementasi. Tim pengembang harus mendefinisikan suatu model siklus hidup dari pengembangan. Setiap anggota pengembang harus memiliki pemahaman yang jelas dari siklus hidup yang diadopsi agar tidak terjadi kesalahan dan kekacauan dalam pengembangan peranti lunak.

2.8 Android

Berdasarkan informasi yang tercatat pada halaman web Android.com, Android merupakan perangkat sistem operasi untuk *mobile phone* yang paling popular di dunia. Dengan Android, ada banyak hal menarik dan unik yang dapat digunakan untuk keperluan sehari-hari, seperti fasilitas aplikasi dari Google, musik, *games,* dan lain-lain. Perangkat Android sudah sangat cerdas dengan fitur-fitur yang tidak dapat ditemukan pada *platform* sistem operasi lainnya.

Meier (2012) mengatakan bahwa Android adalah ekosistem terdiri dari kombinasi tiga komponen, yaitu:

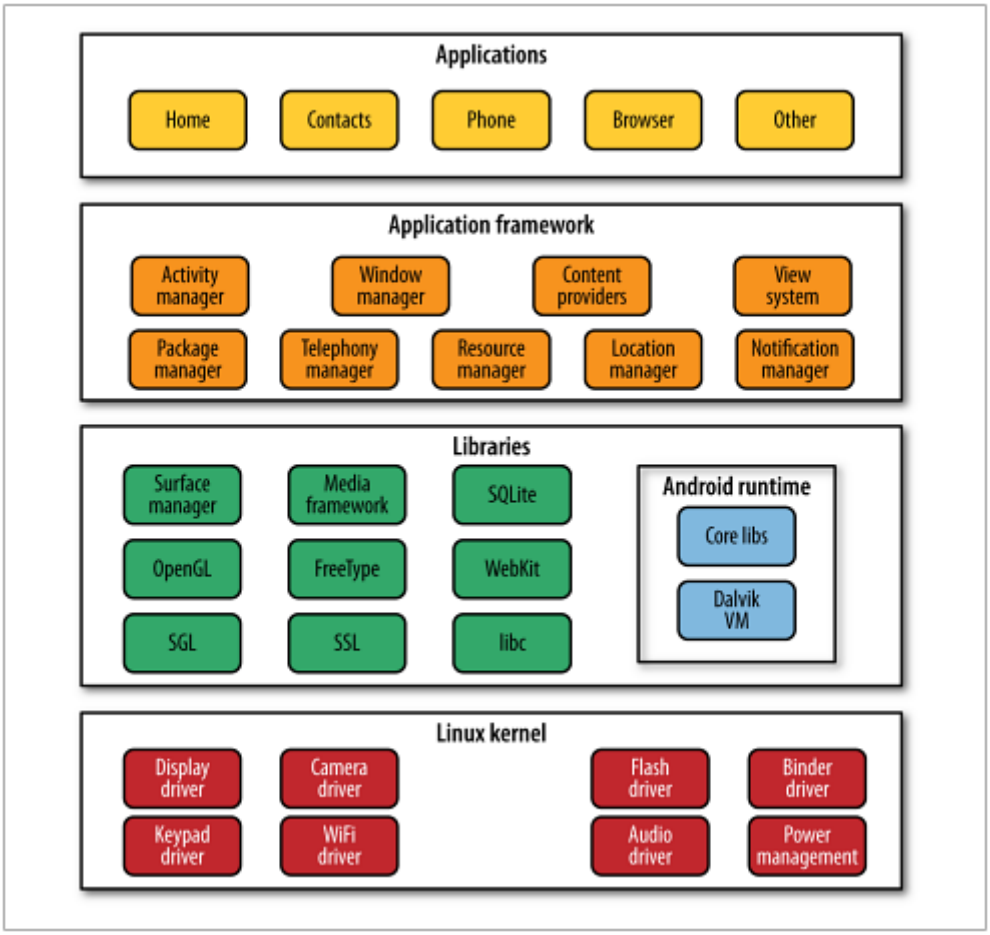
1. Gratis, sistem operasi dengan sumber terbuka (*open-source*) untuk perangkat *embedded.*
2. *Platform* pengembangan yang *open-source* untuk membuat aplikasi.
3. Perangkat, secara detail telepon genggam, yang menjalankan sistem operasi Android dan aplikasi yang dibuat untuknya.

Secara spesifik, Android terdiri dari beberapa bagian yang penting dan saling bergantung satu sama lain, seperti:

1. *Compatibility Definition Document (CDD)* dan *Compatibility Test Suite (CTS)* yang mendeskripsikan kapabilitas yang dibutuhkan sebuah perangkat untuk mendukung *stack* perangkat lunak.
2. Kernel sistem operasi Linux yang menyediakan *low-level interface* dengan perangkat keras, pengaturan memori, dan kontrol proses, semua dioptimalkan untuk perangkat bergerak dan perangkat *embedded.*
3. *Library* sumber terbuka untuk pengembangan aplikasi termasuk SQLite, WebKit, OpenGL, dan *media manager*.
4. *Run Time* untuk menjalankan dan menampung aplikasi Android, termasuk *Dalvik Virtual Machine (DVM)* dan *core libraries* yang menyediakan fungsionalitas spesifik Android. *Run Time* dirancang untuk menjadi kecil dan efisien, sehingga dapat digunakan di perangkat bergerak.
5. *Framework* aplikasi yang secara *agnostically* membuka servis dari sistem kepada *layer* aplikasi, termasuk *window manager*, *location manager*, basis data, telepon, dan sensor.

6. Kumpulan dari aplikasi utama *pre-installed*.
7. Sebuah *Software Development Kit (SDK)* yang digunakan untuk membuat aplikasi, termasuk alat-alat (*tools*) yang berkaitan, *plug-ins*, dan dokumentasi.

Gargenta (2011) mengatakan bahwa Android terdiri dari kernel yang berdasar pada Linux kernel 2.6 dan Linux kernel 3.x (untuk Android versi 4 ke atas). Android dilengkapi dengan *middleware, libraries*, dan API yang ditulis menggunakan Bahasa C dan perangkat lunak aplikasi yang berjalan pada sebuah *framework* aplikasi, termasuk *library* yang cocok dengan Java. Android akan mengeksekusi program di atas *Dalvik Virtual Machines (DVM)* yang akan melakukan kompilasi terhadap *Java byte code*.



Gambar 2.6 Android *stack*
Sumber : Gargenta (2011)