BAB II
LANDASAN TEORI

2.1 Pengenalan Wajah

Pengenalan wajah (*face recognition*) merupakan salah satu teknologi biometrik yang sekarang telah diterapkan untuk banyak aplikasi dalam bidang keamanan, antara lain *Access security system*, *Authentification system*, hingga sebagai alat bantu dalam pelacakan pelaku kriminal. Namun dalam perkembangannya masih terdapat beberapa permasalahan, selain masalah komputasi dan kapasitas penyimpanan data, kondisi citra wajah yang menjadi masukan (*input*) sistem juga merupakan masalah yang penting. Beberapa aspek penting yang mempengaruhi kondisi citra wajah manusia diantaranya adalah pencahayaan, ekspresi wajah dan perubahan atribut seperti kumis, janggut dan kacamata. Sistem pengenalan wajah adalah aplikasi pengolahan citra yang dapat mengidentifikasi atau memverifikasi seseorang melalui citra digital atau frame video (Gunawan, 2012).

Saat ini sistem pengenalan wajah telah menjadi salah satu aplikasi pengolahan citra yang cukup popular terutama di dalam bidang keamanan seperti verifikasi kartu kredit dan identifikasi penjahat. Sistem pengenalan citra wajah umumnya mencakup empat modul utama (LI SZ, 2011).

1. Deteksi wajah (*face detection*).
2. Penyelasaran wajah (*face alignment*)
3. Ekstraksi fitur (*feature extraction*)
4. Pencocokan (*matching*)

2.2 *Principal Component Analysis* (PCA)

Metode PCA atau dikenal juga dengan nama Hotelling Transform adalah sebuah metode yang menghasilkan transformasi orthogonal yang disebut dengan nama eigenimage dimana sebuah citra direpresentasikan ke dalam bentuk proyeksi linier searah dengan eigenimage yang bersesuaian dengan nilai eigen terbesar dari

matriks kovarians. Matriks covarian ini dibangun dari sekumpulan *image training* yang diperoleh dari berbagai objek. Terdapat beberapa metode pengenalan berdasarkan PCA misalnya *eigenface* dan *fisherface*.

2.3 *Eigenface*

Eigenface merupakan metode yang digunakan untuk melakukan ekstraksi ciri wajah. Metode *eigenface* ini pertama kali dikembangkan oleh Matthew Turk dan Alex Pentlad dari *Vision and Modelling Group, The Media Laboratory, Massachusett Institute of Technology* pada tahun 1987. Kemudian metode ini disempurnakan lagi oleh Turk dan Pentlad pada tahun 1991 dengan mengubah cara penghitungan matriks kovarian. Dasar dari metode ini ialah *Principal Component Analysis* (PCA).

Metode PCA melakukan proyeksi dari ruang citra dengan dimensi yang lebih tinggi ke ruang ciri dengan dimensi yang lebih rendah untuk meningkatkan efisiensi dalam proses komputasi dan mengurangi storage yang diperlukan serta dapat memaksimalkan jarak antara semua kelas wajah.

Analisis algoritma *eigenface* pada pengenalan wajah merupakan dasar bagi seseorang yang ingin mendalami ilmu biometrik karena identifikasi dilakukan dengan *pattern matching* sederhana tanpa menggunakan metode pembelajaran khusus. Metode *eigenface* ini banyak digunakan karena memiliki tingkat akurasi yang cukup baik dan sistem komputasi yang tidak terlalu rumit. Berikut merupakan tahapan perhitungan eigenface:

1. Membuat suatu himpunan matriks yang terdiri dari seluruh *training image* yang ada di *database* ($\Gamma_1, \Gamma_2, \ldots \Gamma_{M)}$

$$\mathbf{Im= (\Gamma_1, \Gamma_2, \ldots \Gamma_M)} \qquad \mathbf{(2.1)}$$

Keterangan :

Im = matriks yang berisi nilai keseluruhan data

Γi = data ke-*i*

2. Setelah data diubah ke dalam bentuk matriks satu dimensi, maka akan terbentuk pula suatu matriks besar yang berisi seluruh data referensi.

Kemudian tentukan nilai rata-rata (*mean*) data referensi tersebut dengan menggunakan persamaan :

$$\boldsymbol{\psi} = \frac{\mathbf{1}}{\boldsymbol{M}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{M}} \boldsymbol{\Gamma n} \qquad \textbf{(2.2)}$$

Keterangan :

M = jumlah data referensi

Ψ = nilai rata-rata

Γn = data ke-n

3. Kemudian cari *feature* PCA atau ciri data (Φ*)* dengan mencari selisih antara *Training image (*Γ_i*)* dengan nilai tengah *(*Ψ*),* apabila ditemukan nilainya di bawah nol ganti nilainya dengan nol. Gunakan persamaan sebagai berikut :

$$\boldsymbol{\Phi_i = \Gamma_i - \Psi} \qquad \textbf{(2.3)}$$

Keterangan :

Φ_i = pola hasil ekstraksi data ke-*i*

Γ_i = data ke-*i*

Ψ = nilai rata-rata

4. Setelah rata-rata (*mean*) ditemukan, langkah selanjutnya ialah menghitung nilai matriks kovarian (C) dengan persamaan berikut :

$$\mathbf{C} = \boldsymbol{AA^T} \qquad \textbf{(2.4)}$$

A = $\{\Phi_1, \Phi_2, \dots, \Phi_M\}$ dan A^T merupakan transpose dari matriks A.matriks A adalah matriks yang berisi informasi pola hasil ekstrasi dari seluruh data yang ada.

5. Setelah matriks kovarian ditemukan, hiitung *eigenvalue (λ)* dan *eigenvector (v)* dengan persamaan berikut :

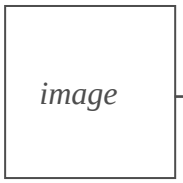
$$\mathbf{AX} = \boldsymbol{\lambda X} \qquad \textbf{(2.5)}$$

Keterangan :

X = vectoreigen

λ = nilai eigen

Nilai eigen dan vectoreigen akan diperoleh dari matriks kovarian yang berisi ciri utama citra. Untuk lebih memahami proses pengenalan citra menggunakan *eigenface,* perhatikan ilustrasi pengenalan berikut :



$$C1=\begin{bmatrix} P11 & P12 & \dots & P1m \\ P21 & P22 & \dots & P2m \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ Pn1 & Pn2 & \dots & Pnm \end{bmatrix}$$

C1 adalah citra referensi ke-1. p adalah nilai pixel citra

↓

Kemudian buat satu himpunan matriks menjadi satu dimensi

C1 = [P11 P12 … P1m P21 P22 … P2m]

C1 merupakan citra referensi ke-1 yang dijadikan matriks satu dimensi.

↓

$$Im = \begin{bmatrix} C1p11 & C1p12 & & C1pnm \\ C2p11 & C2p12 & \dots & C2pnm \\ C3p11 & C3p12 & & C3pnm \\ \vdots & \vdots & \ddots & \vdots \\ Cmp11 & Cmp12 & \cdots & CMpnm \end{bmatrix}$$

M = jumlah citra referensi

↓

Kemudian cari nilai rata-rata (*mean*) citra dengan persamaan :

$$\boldsymbol{\psi = \frac{1}{M} \sum_{n=1}^{M} \Gamma n} \qquad \textbf{(2.6)}$$

Mean = [Mp_{11} Mp_{12} ⋯ M_{pnm}]

↓

Kemudian ekstraksi citra dengan mengurangkan setiap citra referensi terhadap *mean* dengan persamaan :

$$\textbf{Fitur}_n \textbf{ = Cn – mean} \qquad \textbf{(2.7)}$$

$$Im = \begin{bmatrix} C1p11 - Mp11 & C1p12 - Mp12 & & C1pnm - Mpnm \\ C2p11 - Mp11 & C2p12 - Mp12 & \dots & C2pnm - Mpnm \\ C3p11 - Mp11 & C3p12 - Mp12 & & C3pnm - Mpnm \\ \vdots & \vdots & \ddots & \vdots \\ Cmp11 - Mp11 & Cmp12 - Mp12 & \cdots & CMpnm - Mpnm \end{bmatrix}$$

↓

Setelah matriks A ditemukan langkah selanjutnya adalah :

Cari matriks kovarian C dengan cara : **C = Fitur*x* Fitur^{T}**

Kemudian, setelah matriks C ddidapatkan, temukan eigen vector dan eigen value berdasarkan matriks C

[ev eval] = eig (C) **(2.8)**

Selanjutnya temukan eigenface dengan mengalihkan fitur dengan eigenvector :

Eig_f = Fitur ^{T} x ev **(2.9)**

$$\text{Eig\_f} = = \begin{bmatrix} fC11 & fC21 & & fCM1 \\ fC12 & fC22 & \dots & fCM2 \\ fC13 & fC23 & & fCM3 \\ \vdots & \vdots & \ddots & \vdots \\ fC1n & fC2n & \cdots & fCMn \end{bmatrix} \text{ x } \begin{bmatrix} ev1 & \cdots & evm \\ \vdots & \ddots & \vdots \\ evn & \cdots & evnm \end{bmatrix}$$

Tentukan eigenface terpilih sebanyak N. Misalkan N=2 maka :

$$\text{Eig\_f} = \begin{bmatrix} ef11 & \cdots & efM-1m-1 & efMm \\ \vdots & \ddots & \vdots & \vdots \\ ef1n & \cdots & efM-1m-1 & efMnm \end{bmatrix}$$

terpilih

kemudian cari bobot setiap citra referensi dengan cara :

W = fitur *x* Eig_*f*N **(2.10)**

Eig_fN adalah matriks yang berisi eigenface terpilih sebanyak N.

$$\text{W} = \begin{bmatrix} W1 & \cdots & WN \\ \vdots & \ddots & \vdots \\ WM & \cdots & WMN \end{bmatrix}$$

M = jumlah baris, N = jumlah kolom

Setelah bobot dari citra referensi didapatkan, langkah selanjutnya ialah mencari bobot citra yang diujikan. Berikut langkah-langkah pencarian bobot citra yang diujikan :

Suatu citra uji x yang telah dipilih, kemudian lakukan normalisasi terhadap citra tersebut menjadi matriks satu dimensi sebagai berikut :

$$Cx = \begin{bmatrix} P11 & P12 & \dots & P1m \\ P21 & P22 & \dots & P2m \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ Pn1 & Pn2 & \dots & Pnm \end{bmatrix}$$ Cx = [P11 P12 … P1m P21 P22 … Pnm]

Kemudian lakukan ekstraksi citra x dengan cara mengurangkan citra x dengan citra rerata mean :

Fiturx = Cx – mean citra referensi **(2.11)**

Fiturx = [CxP11-Mp1 CxP12-Mp2 … Cxpnm-Mpnm]

Selanjutnya hitung bobot citra uji x dengan cara :

Wx = Fitur *x* Eig_*f*N **(2.12)**

Wx =[Wx1 … Wxn]

Berdasarkan alur proses di atas terlihat bagaimana proses awal sampai ditemukannya bobot dari citra referensi (citra dengan berbagai usia) dan citra uji. Bobot tersebut memberikan informasi ciri utama dari setiap citra.

2.4 Android

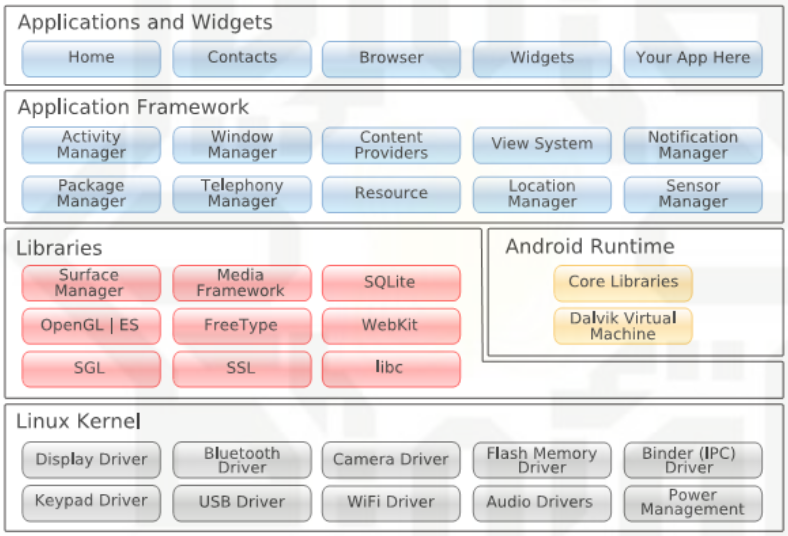
2.4.1 Pengertian Android

Android adalah sebuah sistem operasi untuk perangkat mobile berbasis linux yang mencakup sistem operasi, middleware, dan aplikasi.Android adalah sistem operasi untuk telepon seluler yang berbasis Linux (Nazruddin, 2011). Android menyediakan platform terbuka bagi para pengembang untuk membuat aplikasi mereka sendiri. Pada awalnya dikembangkan oleh Android Inc, sebuah perusahaan pendatang baru yang membuat perangkat lunak untuk ponsel yang kemudian dibeli oleh Google Inc. Untuk pengembangannya, dibentuklah Open

Handset Alliance(OHA), konsorsium dari 34 perusahaan perangkat keras, perangkat lunak, dan telekomunikasi termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan Nvidia.

2.4.2 Struktur Android

Arsitektur Android merupakan sebuah kernel Linux dan sekumpulan pustaka C / C++ dalam suatu *framework* yang menyediakan dan mengatur alur proses aplikasi (Burnette, 2010). Berikut paket sistem operasi Android yang tediri dari beberapa unsur seperti tampak pada gambar :



Gambar 2. 1 Arsitektur Android (Burnette, 2010)

a. Linux Kernel

Android dibangun di atas kernel Linux 2.6. Namun secara keseluruhan Android bukanlah Linux, karena dalam Android tidak terdapat paket standar yang dimiliki oleh Linux lainnya. Linux merupakan sistem operasi terbuka yang handal dalam manajemen memori dan proses. Oleh karenanya pada Android hanya terdapat beberapa servis yang diperlukan seperti keamanan, manajemen memori, manajemen proses, jaringan dan driver. Kernel Linux menyediakan driver layar, kamera, keypad, WiFi, Flash Memory, audio, dan IPC (*Interprocess Communication*) untuk mengatur aplikasi dan lubang keamanan.

b. Libraries

Android menggunakan beberapa paket pustaka yang terdapat pada C/C++ dengan standar *Berkeley Software Distribution* (BSD) hanya setengah dari yang aslinya untuk tertanam pada kernel Linux. Beberapa *library* diantaranya:

1. *Media Library* untuk memutar dan merekam berbagai macam format audio dan video.
2. *Surface Manager* untuk mengatur hak akses layer dari berbagai aplikasi.
3. *Graphic Library* termasuk didalamnya *SGL* dan *OpenGL*, untuk tampilan 2D dan 3D.
4. *SQLite* untuk mengatur relasi *database* yang digunakan pada aplikasi.
5. *SSl* dan *WebKit* untuk browser dan keamanan internet.
6. *Library* tersebut bukanlah aplikasi yang berjalan sendiri, namun hanya dapat digunakan oleh program yang berada di level atasnya. Sejak versi Android 1.5, pengembang dapat membuat dan menggunakan *library* sendiri menggunakan *Native Development Toolkit* (NDK).

c. Android Runtime

Pada Android tertanam paket *library* inti yang menyediakan sebagian besar fungsi Android. Inilah yang membedakan Android dibandingkan dengan sistem operasi lain yang juga mengimplementasikan Linux. *Android Runtime* merupakan mesin virtual yang membuat aplikasi Android menjadi lebih tangguh dengan paket *library* yang telah ada. Dalam Android Runtime terdapat 2 bagian utama, diantaranya :

1. *Library* Inti, Android dikembangkan melalui bahasa pemrograman Java, tapi Android Runtime bukanlah mesin virtual Java. Library inti Android menyediakan hampir semua fungsi yang terdapat pada *library* Java serta beberapa *library* khusus Android.
2. Mesin Virtual Dalvik, Dalvik merupakan sebuah mesin virtual yang dikembangkan oleh Dan Bornstein yang terinspirasi dari nama sebuah perkampungan yang berada di Iceland. Dalvik hanyalah interpreter mesin virtual yang mengeksekusi file dalam format *Dalvik Executable* (*.dex). Dengan format ini Dalvik akan mengoptimalkan efisiensi penyimpanan

dan pengalamatan memori pada file yang dieksekusi. Dalvik berjalan di atas kernel Linux 2.6, dengan fungsi dasar seperti *threading* dan manajemen memori yang terbatas.

d. Application Framework

Kerangka aplikasi menyediakan kelas-kelas yang dapat digunakan untuk mengembangkan aplikasi Android. Selain itu, juga menyediakan abstraksi generic untuk mengakses perangkat, serta mengatur tampilan *user interface* dan sumber daya aplikasi. Bagian terpenting dalam kerangka aplikasi Android adalah sebagai berikut (Burnette, 2010)*:*

1. *Activity Manager*, berfungsi untuk mengontrol siklus hidup aplikasi dan menjaga keadaan ”*Backstack“* untuk navigasi penggunaan.
2. *Content Providers*, berfungsi untuk merangkum data yang memungkinkan digunakan oleh aplikasi lainnya, seperti daftar nama.
3. *Resource Manager*, untuk mengatur sumber daya yang ada dalam program. Serta menyediakan akses sumber daya diluar kode program, seperti karakter, grafik, dan file layout.
4. *Location Manager*, berfungsi untuk memberikan informasi detail mengenai lokasi perangkat Android berada.
5. *Notification Manager*, mencakup berbagai macam peringatan seperti, pesan masuk, janji, dan lain sebagainya yang akan ditampilkan pada *status bar*.

2.4.3 Kelebihan Android

Saat ini sudah banyak *mobile platform* yang tersedia di pasaran, termasuk didalamnya Symbian, iPhone, Windows Mobile, BlackBerry, Java Mobile Edition, Linux Mobile (LiM0), dan banyak lagi. Namun ada beberapa hal yang menjadi kelebihan Android. Walaupun beberapa fitur-fitur yang ada telah muncul sebelumnya pada platform lain, Android adalah yang pertama menggabungkan hal seperti berikut (Burnette, 2010)*:*

1. Keterbukaan (*Open Source*), Bebas pengembangan tanpa dikenakan biaya terhadap sistem karena berbasiskan Linux dan *open source*. Pembuat

perangkat menyukai hal ini karena dapat membangun *platform* yang sesuai yang diinginkan tanpa harus membayar *royalty*. Sementara pengembang *software* menyukai karena Android dapat digunakan diperangkat manapun dan tanpa terikat oleh vendor manapun.

2. Arsitektur komponen dasar Android terinspirasi dari teknologi internet *Mashup*. Bagian dalam sebuah aplikasi dapat digunakan oleh aplikasi lainnya, bahkan dapat diganti dengan komponen lain yang sesuai dengan aplikasi yang dikembangkan.
3. Banyak dukungan service, kemudahan dalam menggunakan berbagai macam layanan pada aplikasi seperti penggunaan layanan pencarian lokasi, *database* SQL, browser dan penggunaan peta. Semua itu sudah tertanam pada Android sehingga memudahkan dalam pengembangan aplikasi.
4. Siklus hidup aplikasi diatur secara otomatis, setiap program terjaga antara satu sama lain oleh berbagai lapisan keamanan, sehingga kerja sistem menjadi lebih stabil. Pengguna tak perlu kawatir dalam menggunakan aplikasi pada perangkat yang memorinya terbatas.
5. Dukungan grafis dan suara terbaik, dengan adanya dukungan 2D grafis dan animasi yang diilhami oleh *Flash* menyatu dalam 3D menggunakan *OpenGL* memungkinkan membuat aplikasi maupun game yang berbeda.
6. Portabilitas aplikasi, aplikasi dapat digunakan pada perangkat yang ada saat ini maupun yang akan datang. Semua program ditulis dengan menggunakan bahasa pemrograman Java dan dieksekusi oleh mesin virtual Dalvik, sehingga kode program portabel antara ARM, X86, dan arsitektur lainnya. Sama halnya dengan dukungan masukan seperti penggunaan *Keyboard,* layar sentuh, *trackball* dan resolusi layar semua dapat disesuaikan dengan program.

2.5 UML

Unified Modelling Language(UML) adalah sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan

mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. (A.S, 2014).

2.5.1 Objek

Objek atau Benda merupakan bagian paling statik dari sebuah model, yang menjelaskan elemen–elemen lainnya dari sebuah konsep. Bentuk dari beberapa objek :

a. *Classes*, sekelompok dari object yang mempunyai atribute, operasi, dan hubungan yang semantik.
b. *Interfaces*, antar-muka yang menghubungkan dan melayani antarkelas dan atau elemen dan mendefinisikan sebuah kelompok dari spesifikasi pengoperasian.
c. *Collaboration*, interaksi dari sebuah kumpulan kelas–kelas atau elemen–elemen yang bekerja secara bersama–sama.
d. *Use cases*, pembentuk tingkah laku objek dalam sebuah model serta di realisasikan oleh sebuah collaboration.
e. *Nodes*, bentuk fisik dari elemen–elemen yang ada pada saat dijalankannya sebuah system.

2.5.2 Relasi

Ada 4 macam hubungan dalam penggunaan UML, yaitu :

a. Dependency, hubungan semantik antara dua objek yang mana sebuah objek berubah mengakibatkan objek satunya akan berubah pula.
b. Association, hubungan antar benda secara struktural yang terhubung diantara objek dalam kesatuan objek.
c. Generalizations, hubungan khusus dalam objek anak yang menggantikan objek induk dan memberikan pengaruhnya dalam hal struktur dan tingkah lakunya kepada objek induk
d. Realizations, hubungan semantik antarpengelompokkan yang menjamin adanya ikatan diantaranya yang diwujudkan diantara *interface* dan kelas atau elements, serta antara *use cases* dan *collaborations*.

2.5.3 Bagan atau Diagram

Diagram adalah yang menggambarkan permasalahan maupun solusi dari permasalahan suatu model. Diagram tersebut (A.S, 2014), adalah sebagai berikut:

1. Usecase Diagram

Usecase adalah abstraksi dari interaksi antara *system* dan *actor*. Use case bekerja dengan cara mendeskripsikan tipe interaksi antara *user* sebuah system dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah system dipakai. Diagram Use Case berguna dalam tiga hal :

a. Menjelaskan fasilitas yang ada (*requirement*)
b. Komunikasi dengan klien
c. Membuat test dari kasus-kasus secara umum

2. *Activity* Diagram

Activity diagram menyediakan analis dengan kemampuan untuk memodelkan proses dalam suatu sistem informasi. *Activity* diagram dapat digunakan untuk alur kerja model, use case individual, atau logika keputusan yang terkandung dalam metode individual. *Activity* diagram juga menyediakan pendekatan untuk proses pemodelan paralel. Pada dasarnya, diagram aktifitas canggih dan merupakan diagram aliran data yang terbaru. Secara teknis, diagram aktivitas menggabungkan ide-ide proses pemodelan dengan teknik yang berbeda termasuk model acara, *statecharts*, dan *Petri Nets*.

3. *Class* Diagram

Diagram kelas, memberikan pandangan secara luas dari suatu sistem dengan menunjukan kelas–kelasnya dan hubungan mereka. Diagram Class mempunyai 3 macam relationalships (hubungan), sebagai berikut:

a. *Association*, suatu hubungan antara bagian dari dua kelas yang terjadi jika salah satu bagian dari kelas mengetahui kelas yang lain dalam melakukan suatu kegiatan.

b. *Aggregation*, hubungan association dimana salah satu kelasnya merupakan bagian dari suatu kumpulan dan memiliki titik pusat yang mencakup keseluruhan bagian.

c. *Generalization*, hubungan turunan dengan mengasumsikan satu kelas merupakan suatu kelas super dari kelas yang lain.

4. *Sequence* Diagram

Sequence diagram menjelaskan interaksi objek yang disusun berdasarkan urutan waktu. Secara mudahnya sequence diagram adalah gambaran tahap demi tahap yang seharusnya dilakukan untuk menghasilkan sesuatu sesuai dengan *use case* diagram.

2.6 Android Studio

Android Studio adalah lingkungan pengembangan baru dan terintegrasi penuh, yang baru saja dirilis oleh Google untuk sistem operasi Android. Android Studio dirancang untuk menjadi peralatan baru dalam pengembangan aplikasi dan juga memberi alternatif lain selain Eclipse yang saat ini menjadi IDE yang paling banyak dipakai.

Saat memulai proyek baru dengan Android Studio, struktur proyek akan muncul bersama dengan hampir semua berkas yang ada di dalam direktori SDK, peralihan ke sistem manajemen berbasis Gradle ini memberikan fleksibilitas yang lebih besar pada proses pembangunanya.

Android Studio mengijinkan untuk melihat perubahan visual apapun yang lakukan pada aplikasi secara langsung. *User* juga bisa melihat perbedaannya jika dipasang pada beberapa perangkat Android berbeda, termasuk konfigurasi dan resolusinya secara bersamaan.

Fitur lain di Android Studio adalah alat-alat baru untuk mengepak dan memberi label kode. Dengan begitu mengijinkan *user* tetap menjadi yang teratas ketika berurusan dengan banyak kode. Program ini juga memakai sistem seret dan jatuhkan untuk memindahkan komponen melalui antar muka pengguna.

Selain itu, lingkungan baru ini juga mendukung *Google Cloud Messaging*. Sebuah fitur yang mengijinkan *user* untuk mengirim data dari server ke perangkat Android *user* melalui cloud, cara terbaik untuk mengirim Pengingat pada apps anda.

Program ini juga membantu *user* untuk melokalisasi aplikasi *user*, memberi *user* gambaran visual untuk tetap memprogram sambil mengontrol alur dari aplikasi.

Kelebihan lain Android Studio:

a. Lingkungan pengembangan yang mantap dan bersifat langsung.
b. Cara termudah untuk menguji performa pada perangkat dengan tipe lain.
c. *Wizard* dan *template* berisi elemen-elemen umum yang ada di semua pemrograman Android.
d. *Editor* dengan fitur lengkap dengan banyak peralatan ekstra untuk mempercepat pengembangan aplikasi *user*.

2.7 Java

Bahasa Java adalah bahasa pemograman yang bersifat terditribusi, multiplatform dan multithread yang berorientasi objek (Object Oriented) yang dimodelkan seperti bahasa C++ namun dengan beberapa keunggulan.

Sejarah Java berawal pada tahun 1991 ketika perusahaan Sun Microsystem memulai Green Project, yakni proyek penilitian untuk membuat bahasa yang akan digunakan pada chip-chip embedded untuk device intelligent consumer electronic.

Karena pada awalnya ditujukan untuk pemograman device kecil, Java memiliki karakteristik berukuran kecil, efisien dan portable ntuk berbgai hardware(Hermawan, 2014).

2.8 Penelitian Terkait

Berikut adalah penilitian terkait dengan penilitian yang dilakukan peniliti:

1. Penilitian yang dilakukan oleh (Oka Sudana et al., 2014) dengan judul sistem pengenalan wajah di android menggunakan metode *eigenface*. Penilitian tersebut menggunakan sebanyak 50 sampel gambar wajah sebagai gambar uji. Gambar uji dan gambar latih diambil menggunakan kamera perangkat android dengan jarak 60 cm. Hasil penelitian tersebut menghasilkan kesimpulan bahwa pengenalan wajah dengan metode

eigenface memberikan akurasi yang baik dengan *succes rate* 94,48% dengan FMR = 2,52% dan FNMR = 3%.

2. Penilitian oleh (Ozdil, 2014) dengan judul *A Survey on Comparison of Face Recognition Algorithm.* Penilitian membahas tentang perbandingan algoritma LBP, *Eigenface, Fisherface*. Menggunakan Metode yang sama, yakni metode eigenface penilitian mencatat tingkat keakuratan 93% dengan tanpa mengorbankan banyak performa prosesor baik itu Intel maupun Arm.
3. Penilitian oleh (Arliani, 2011) dengan judul Analisis pengenalan wajah hasil akuisisi webcam menggunakan metode *eigenface* dan *support vector machine*. Penilitian ini bertujuan untuk membuat sistem yang dapat mengenali citra wajah yang diuji. Menggunakan 150 total gambar wajah. 100 untuk pengujian 50 untuk pelatihan. Penilitian ini menggunakan metode eigenface dan support vector machine (SVM). Dari hasil pengujian, didapatkan hasil terbaik dengan akurasi tertinggi yaitu 95%.
4. Penilitian yang dilakukan oleh (Urifan, 2010) dengan judul Pengenalan Wajah Dengan Metode Eigenface. Penilitian membahas tentang penggunaan organ tubuh berupa wajah manusia untuk indikasi pengenalan seseorang atau *face recognition.* Penilitian ini menunnjukkan ukuran citra mempengaruhi jarak Euclidian yang mempengaruhi akurasi pengenalan.
5. Penilitian oleh (Bayu, Hendriawan, 2009) yang berjudul Penerapan Face Recognition Dengan Metode *Eigenface* Dalam Intelligent *Home* Security. Hasil penilitian ini menunjukkan akurasi pengenalan sekitar 86,5%