BAB 2
TINJAUAN PUSTAKA

2.1 Studi Literatur

Penelitian mengenai *face recognition* menggunakan metode *eigenface* pernah dilakukan oleh Liton Chandra Paul. Kesimpulan jurnal tesis Liton adalah sistem berhasil mengenali wajah manusia dengan berbagai orientasi. Namun dalam penelitian tersebut tidak dijelaskan bagaimana sistem mengambil wajah dari camera yang digunakan sehingga hanya bagian wajah saja yang terdeteksi[8]. Sedangkan dalam penelitian Mariappan dkk algoritma dalam proses *face detection* adalah *image threshold, filtering, edge detection, morphological* dan *circle detection* (Mariappan, 2014). Hasil dari penelitian tersebut sistem bekerja dengan baik untuk latar belakang gambar yang berbeda-beda. Wajah dapat dideteksi bahkan jika orang tersebut berada pada jarak yang berbeda-beda dari kamera[9].

Kemudian dalam penelitian Figen disimpulkan bahwa untuk meningkatkan tingkat keberhasilan pengenalan wajah menggunakan metode *eigenface*, dapat diperkaya dengan informasi segitiga wajah dan memperbesar *database*[10]. Sedangkan Malkauthekar mengatakan dalam hasil penelitiannya bahwa metode *manhatan distance* lebih baik daripada *euclidean distance*[11]. Peneliti Pooja mengatakan *mahalanobis distance* mempunyai kinerja lebih baik dari pada *euclidean distance* dalam hal *face recognition*[12].

Kemudian Fan dkk dalam penelitiannya mengatakan bahwa fitur yang sangat popular dalam *machine learning* dan *patern recognition* adalah *Principal Component Analysis (PCA)*[13]. Sedangkan sistem pengenalan wajah atau *face recognition* terdiri dari 2 bagian. Pertama adalah bagian *hardware* sedangkan yang ke dua adalah *software. Hardware* atau perangkat keras yang digunakan adalah kamera, laptop dan mikrokontroller. *Software* atau perangkat lunak yang digunakan adalah *LabVIEW (Laboratory Virtual Instrument Engineering Workbench)*.

2.2 Tinjauan Teori

2.2.1 Pengertian Pengolahan Citra Digital

Pengolahan citra digital adalah pemerosesan gambar berdimensi dua melalui komputer digital (Jain, 1989). Pengolahan citra digital adalah istilah umum untuk berbagai teknik yang keberadaanya untuk memanipulasi dan memodifikasi citra dengan berbagai cara (Efford, 2000). Pengolahan citra merupakan bagian penting yang mendasari berbagai aplikasi nyata (Khadir,

2013). Aplikasi pengolahan citra seperti pengenalan pola, penginderaan jarak jauh melalui satelit dan *machine vision* (Susanto, 2013).

2.2.2 Citra Berwarna

Citra berwarna merupakan jenis citra yang menyajikan warna dalam bentuk komponen R *(red)*, G *(green)*, dan B *(blue)*. Setiap komponen warna menggunakan 8 bit (nilainya berkisar antara 0 sampai dengan 255). Jumlah variasi warna yang dihasilkan adalah 255 x 255 x 255 atau 16.581.375 warna[14]. Tabel 2.1 menunjukan contoh warna dan nilai R, G dan B.

Tabel 2.1 Warna dan nilai penyusun warna

Warna	R	G	B
Merah	255	0	0
Hijau	0	255	0
Biru	0	0	255
Hitam	0	0	0
Putih	255	255	255
Kuning	0	255	255

2.2.3 Citra Berskala Keabuan

Citra jenis ini menangani gradasi warna hitam dan putih (Khadir, 2013). Citra ini menghasilkan efek warna abu-abu. Pada jenis gambar ini warna dinyatakan dengan intensitas. Intensitas berkisar antara 0 sampai 255. Nilai 0 menyatakan hitam dan 255 menyatakan putih[14].

2.2.4 Citra Biner

Citra biner dinyatakan dengan sebuah nilai 0 dan 1 (Khadir, 2013). Nilai 0 menyatakan nilai hitam dan 1 menyatakan nilai putih. Citra jenis ini banyak dipakai dalam pemrosesan citra. Contoh aplikasi citra biner adalah mendeteksi tepi objek suatu citra[14].

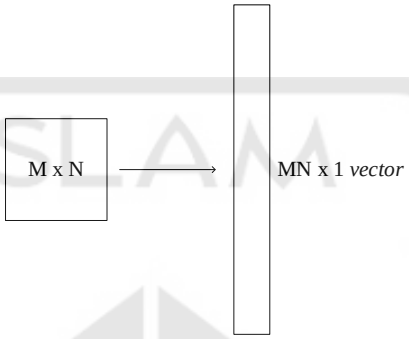
2.2.5 Metode *Eigenface*

Eigenface adalah kumpulan dari *eigen vector* yang berasal dari kovarian matrik dengan distribusi acak pada citra wajah dimensi tinggi. Metode ini mentransformasikan citra wajah ke dalam sebuah kumpulan karakteristik fitur citra yang dinamakan *eigenface*[15]. *Eigenface* didapatkan dengan mengkombinasikan *eigen vector* dengan citra sesungguhnya[16]. Metode *eigenface* mempunyai komputasi yang sederhana dan cepat dibandingkan dengan penggunaan metode yang memerlukan banyak pembelajaran seperti jaringan syaraf tiruan. Metode *eigenface* berfungsi untuk menghitung *eigenvalue* dan *eigenvector* yang akan digunakan sebagai fitur dalam melakukan pengenalan[17].

2.2.6 Algoritma *Eigenface*

Asumsikan bahwa setiap citra $\Gamma(x, y)$ memiliki resolusi M x N *pixel*. Kemudian ubah setiap citra menjadi citra abu-abu (*grayscale*), setelah itu mengubah citra ke dalam bentuk matrik. Sehingga secara umum algoritma *eigenface* adalah sebagai berikut:

Mengubah matrik citra abu-abu berukuran M x N menjadi citra matrik berukuran MN x 1 seperti Gambar 2.1 berikut:



Gambar 2.1 Mengubah citra resolusi M x N *pixel* menjadi MN x 1 *vector*

Setiap citra dinyatakan Γ_i. Γ_i adalah *vector* yang mereprensentasikan citra wajah. Γ_i adalah *vector* berukuran MN x 1 yang berkorespondensi dengan citra wajah M x N *pixel*. Kemudian membentuk *training set* citra wajah $(\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, \dots \dots \dots \Gamma_m)$ untuk pelatihan. Citra wajah harus mempunyai resolusi serta *align* yang sama.

Menghitung *average face vector* (Ψ) dengan persamaan berikut:

$$\Psi = \frac{1}{n} \sum_{i=1}^{n} \Gamma_n \tag{2.1}$$

dengan nilai:

Ψ = *average face vector* / rata-rata *vector* citra wajah

Γ_n = *vector* citra wajah ke – n ukuran MN x 1

n = jumlah citra wajah

Menghitung selisih antara *vector* wajah Γ_i dengan *average face vector* Ψ. Selisih tersebut di simbolkan Φ.

$$\Phi_i = \Gamma_i - \Psi \tag{2.2}$$

dengan nilai:

Φ_i = nilai ke-i selisih antara *vector* wajah (Γ_i) dengan *average face vector* (Ψ)

Menghitung matrik kovarian C sebagai berikut:

$$C = \frac{1}{n} \sum_{i=1}^{n} \Phi_i \Phi_i^T = AA^T \text{ (MN x MN)} \tag{2.3}$$

dengan nilai:

C = matrik kovarian

A = $[\phi_1\ \Phi_2\ \Phi_3\ \Phi_4\ \Phi_5, \ldots \ldots \Phi_n]$ matrik MN x M

A^T = matrik *transpose* dari matrik A

Menghitung *eigenvector* (u_i) dari matrik kovarian C. Berdasarkan Persamaan (2.3) matrik kovarian C memiliki dimensi MN x MN. Ukuran tersebut sangat besar sehingga membuat keadaan menjadi tidak praktis[18]. Misal citra dengan resolusi 256 x 256 *pixel* maka proses komputasi akan menghitung matrik 65.356 x 65.356 dan 65.356 *eigenvalue*. Nilai tersebut merupakan jumlah data yang sangat besar dan tidak efisien untuk proses komputasi. Sehingga diperlukan beberapa trik untuk menghitung *eigenvaluenya* (B.Utomo, 2011). Solusi untuk menghitung *eigenvector* (u_i) dari matrik AA^T yang memiliki ukuran yang sangat besar adalah sebagai berikut:

Menghitung matrik L menggunakan Persamaan (2.4) berikut:

$$L = A^T A \tag{2.4}$$

Menghitung *eigenvector* (v_i) dari matrik L. Matrik L memiliki resolusi M x M, kemudian menghitung *eigenvector* (v_i) dari $A^T A$ dengan menyelesaikan persamaan berikut:

$$A^T A v_i = \mu_i v_i \tag{2.5}$$

dengan nilai:

v_i = *eigenvector* dari matrik L

μ_i = *eigenvector* dari matrik kovarian C

Kesimpulannya untuk menemukan *eigenvector* dan *eigenvalue* dari matrik kovarian C cukup dengan mencari *eigenvector* dan *eigenvalue* dari matrik L.

Mencari *eigenvector* v_i dari matrik L dengan *eigenvalue* $\neq 0$.

Mencari M *eigenface* dari matrik L dengan persamaan berikut:

$$u_i = \sum_{i=1}^{n} v_i \Phi_i \tag{2.6}$$

dengan nilai:

u_i = nilai *eigenface* ke - i

Memproyeksikan citra *dataset* pada dasarnya adalah mencari bobot atau nilai kombinasi linier terbaik untuk merepresentasikan wajah (B.Utomo, 2011). Kombinasi linier dapat dihitung dengan mengalikan matrik selisih antar citra (Φ) dengan nilai *eigenface* (u_i). nilai bobot(ω) dapat dinyatakan sebagai persamaan berikut:

$$\omega_i = u_i^T \Phi_i \tag{2.7}$$

$$\omega_i = u_i^T (\Gamma_i - \Psi) \tag{2.8}$$

dengan nilai:

ω_i = nilai bobot ke - i

Sehingga matrik bobot (Ω) dapat dirumuskan sebagai berikut:

$$\Omega_i^T = [\omega_1, \omega_2, \omega_3 \dots \dots \dots \omega_K] \quad (2.9)$$

dengan nilai:

Ω_i^T = *transpose* matrik bobot

2.2.7 Proyeksi Citra *Datatest*

Citra *datatest* adalah citra keluaran dari *face detection*. Memproyeksikan citra *datatest* adalah mengubah citra *datatest* M x N menjadi citra *datatest* MN x 1. *Vector* citra *datatest* tersebut disimbolkan Γ_{test}. Sehingga persamaan yang digunakan untuk meproyeksikan citra *datatest* adalah sebagai berikut:

$$\omega_{test} = u_i^T(\Gamma_{test} - \Psi) \quad (2.10)$$

$$\Omega_{test}^T = [\omega_1, \omega_2, \omega_3 \dots \dots \dots \omega_K] \quad (2.11)$$

dengan nilai:

ω_{test} = nilai bobot citra *datatest*

Ω_{test}^T = *transpose* matrik bobot citra *datatest*

Γ_{test} = *vector* citra *datatest* berukuran MN x 1

2.2.8 Aplikasi Citra Menggunakan *Euclidean Distance*

Jarak *euclidean* adalah salah satu metode yang digunakan pada aplikasi temu kembali citra. metode jarak *euclidean* digunakan untuk mencari jarak dengan data fitur yang telah didapat. Data fitur pada penelitian ini adalah Ω_i dan Ω_{test} yang telah dijelaskan pada sub bab 2.2.6 sampai 2.2.7. kemudian hasilnya diambil dari jarak terkecil yang didapat.

Euclidean Distance dihitung dengan persamaan berikut:

$$d(\Omega_{test}, \Omega_i) = \sqrt{\sum_{i=1}^{n} (\Omega_{test} - \Omega_i)^2} \quad (2.12)$$

dengan nilai:

$d(\Omega_{test}, \Omega_i)$ = nilai jarak *euclidean*

Hasil dari penelitian ini akan diukur dengan menghitung nilai *error* dari hasil klasifikasi yang benar dengan semua data uji[19]. Tingkat akurasi ini dihitung dengan menggunakan rumus :

$$Error = \frac{\text{jumlah klasifikasi yang benar}}{\text{jumlah semua data uji}} \text{ x } 100\% \quad (2.13)$$