

LAPORAN TUGAS BESAR II
IF2123 ALJABAR LINIER DAN GEOMETRI

**APLIKASI KONSEP *EIGENVALUE* DAN *EIGENFACE* PADA
TEKNOLOGI PENGENALAN WAJAH (*FACE RECOGNITION*)**



Kelompok:

in Bill we trust

Anggota:

Jeffrey Chow	13521046
Bill Clinton	13521064
Chiquita Ahsanunnisa	13521129

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

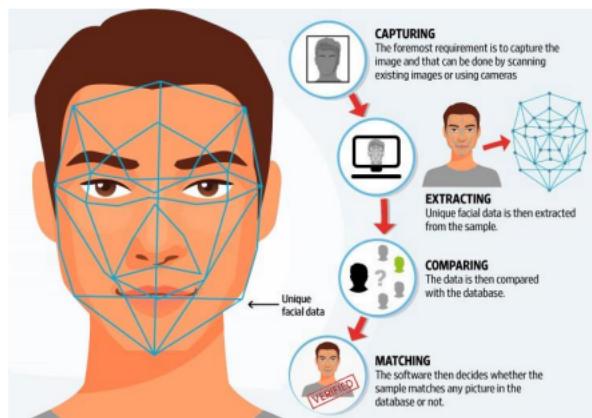
DAFTAR ISI	i
BAB I DESKRIPSI MASALAH	1
BAB II TEORI SINGKAT	2
2.1 <i>Eigenvalue</i>	2
2.2 <i>Eigenvector</i>	2
2.3 Perkalian Matriks	2
2.4 Dekomposisi QR.....	3
2.5 Metode Gram-Schmidt.....	3
2.6 Perhitungan <i>Eigenvalue</i> dan <i>Eigenvector</i>	3
2.7 Matriks Kovarian	4
2.8 <i>Euclidean Distance</i>	4
2.9 <i>Image Normalization</i>	4
BAB III IMPLEMENTASI PROGRAM	6
3.1 Algoritma.....	6
3.1.1 Tahap <i>Image Preprocessing</i>	6
3.1.2 Tahap <i>Training</i>	6
3.1.3 Tahap <i>Testing</i>	8
3.2 <i>Tech Stack</i>	9
3.3 Program	10
3.3.1 Modul dan File	10
3.3.2 Modul configImages	10
3.3.3 Modul function.....	10
3.3.4 GUI dan Program Utama	11
BAB IV EKSPERIMEN.....	13
4.1 Masukan <i>Testing Image</i> dari File	13
4.2 Masukan <i>Testing Image</i> dari Kamera.....	14

4.3	Analisis.....	15
BAB V KESIMPULAN, SARAN, DAN REFLEKSI		16
5.1	Kesimpulan.....	16
5.2	Saran	16
5.3	Refleksi.....	16
REFERENSI		17
LAMPIRAN		18

BAB I

DESKRIPSI MASALAH

Face Recognition atau pengenalan wajah merupakan teknologi biometrik yang dapat dipakai untuk mengidentifikasi wajah manusia. *Face recognition* dapat digunakan untuk berbagai kepentingan, khususnya keamanan, misalnya saja fitur *Face ID*. Program pengenalan wajah melibatkan berbagai citra wajah yang telah disimpan dalam *database* yang kemudian berdasarkan kumpulan wajah tersebut, program dapat mempelajari bentuk wajah lalu mencocokkan citra yang akan diidentifikasi dengan kumpulan citra wajah yang sudah dipelajari. Alur untuk sistem pengenalan wajah terlihat pada Gambar 1 berikut.



Gambar 1. Alur proses di dalam sistem pengenalan wajah (Sumber:
<https://www.shadowsystem.com/page/20>)

Ada berbagai metode yang dapat digunakan untuk memeriksa citra wajah dari kumpulan citra yang telah diketahui, misalnya *cosine similarity* dan jarak Euclidean, *principal component analysis* (PCA), serta *eigenface*. Pada tugas besar II kali ini, akan dibuat program pengenalan wajah menggunakan *eigenface*.

Kumpulan citra wajah akan digunakan dengan representasi matriks. Dari representasi tersebut, akan dicari matriks *eigenface*. Program pengenalan wajah dapat dibagi menjadi dua tahap berbeda, yakni tahap *training* dan pencocokan. Pada tahap *training*, diberikan kumpulan *dataset* berupa citra wajah. Citra tersebut akan dinormalisasi dari RGB (*Red, Green, and Blue*) ke *Grayscale* (matriks). Hasil normalisasi tersebut akan digunakan dalam perhitungan *eigenface*. Perhitungan ini melibatkan *eigenvalue* atau nilai eigen dan *eigenvector* atau vektor eigen.

BAB II

TEORI SINGKAT

2.1 *Eigenvalue*

“Eigen” adalah kata yang berasal dari bahasa Jerman yang memiliki arti yaitu karakteristik. Definisi nilai eigen sendiri berhubungan dengan transformasi linear. Nilai eigen atau *eigenvalue* menyatakan nilai karakteristik dari sebuah matriks persegi berukuran $n \times n$. *Eigenvalue* merupakan nilai skalar yang digunakan untuk mentransformasi sebuah vektor eigen atau *eigenvector*.

2.2 *Eigenvector*

Jika ada suatu matriks A berukuran $n \times n$, maka vektor tidak nol di \mathbb{R}^n adalah vektor eigen atau *eigenvector* dari matriks A jika

$$Ax = \lambda x$$

dengan λ adalah *eigenvalue* dari matriks A dan x adalah eigenvector yang bersesuaian dengan λ . Eigenvector ini menyatakan matriks kolom yang apabila dikalikan dengan sebuah matriks persegi akan menghasilkan vektor yang merupakan kelipatan dari vektor itu sendiri.

2.3 Perkalian Matriks

Misalkan ada matriks A berukuran $m \times r$ dan matriks B berukuran $r \times n$. Jika kedua matriks tersebut dikalikan, akan dihasilkan matriks berukuran $m \times n$. Perkalian dua matriks ini memiliki syarat yaitu untuk perkalian $A \times B$, jumlah kolom A harus sama dengan jumlah baris B . Algoritma untuk melakukan dua buah matriks adalah sebagai berikut.

Perkalian matriks : $C_{m \times n} = A_{m \times r} \times B_{r \times n}$

Algoritma :

```
for i←1 to m do
    for j←1 to n do
        cij ← 0
        for k←1 to r do
            cij ← cij + aik * bkj
        end for
    end for
end for
```

2.4 Dekomposisi QR

Dekomposisi matriks merupakan proses memfaktorkan sebuah matriks menjadi hasil kali dari beberapa matriks lain. Ada berbagai metode dalam mendekomposisi matriks dan salah satunya adalah menggunakan metode *QR decomposition* atau dekomposisi QR. Metode dekomposisi QR adalah metode untuk mendekomposisi sebuah matriks persegi dengan memfaktorkannya menjadi matriks Q dan R, dengan Q adalah matriks ortogonal, yaitu matriks yang memenuhi syarat $Q^T Q = I$, dan R adalah matriks segitiga atas, yaitu matriks persegi yang elemen-elemen di bawah diagonal utamanya bernilai nol. Ada berbagai metode yang dapat digunakan untuk mencari melakukan dekomposisi ini, namun metode yang kami gunakan untuk tugas besar kali ini adalah metode Gram-Schmidt.

2.5 Metode Gram-Schmidt

Misal ada suatu matriks persegi A yang dapat ditulis sebagai berikut.

$$A = \begin{bmatrix} \mathbf{a}_1 & | & \mathbf{a}_2 & | & \cdots & | & \mathbf{a}_n \end{bmatrix}.$$

Setelah itu,

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \\ \mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\mathbf{e}_1, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}. \\ \mathbf{u}_{k+1} &= \mathbf{a}_{k+1} - (\mathbf{a}_{k+1} \cdot \mathbf{e}_1)\mathbf{e}_1 - \cdots - (\mathbf{a}_{k+1} \cdot \mathbf{e}_k)\mathbf{e}_k, & \mathbf{e}_{k+1} &= \frac{\mathbf{u}_{k+1}}{\|\mathbf{u}_{k+1}\|}. \end{aligned}$$

Dengan demikian, matriks A dapat didekomposisi menjadi matriks Q dan R dengan ketentuan sebagai berikut.

$$A = \begin{bmatrix} \mathbf{a}_1 & | & \mathbf{a}_2 & | & \cdots & | & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 & | & \mathbf{e}_2 & | & \cdots & | & \mathbf{e}_n \end{bmatrix} \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{e}_1 & \mathbf{a}_2 \cdot \mathbf{e}_1 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_1 \\ 0 & \mathbf{a}_2 \cdot \mathbf{e}_2 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_n \end{bmatrix} = QR.$$

2.6 Perhitungan *Eigenvalue* dan *Eigenvector*

Metode yang kami gunakan untuk menghitung *eigenvalue* dan *eigenvector* adalah dengan menggunakan *power iteration* dan *QR decomposition*. *Power iteration* sendiri adalah metode aljabar linier untuk memperkirakan *eigenvalue* dan *eigenvector* dari suatu matriks simetris. Untuk mendapatkan *eigenvalue*, dilakukan proses dekomposisi QR pada matriks untuk menemukan matriks Q lalu matriks awal diganti dengan hasil dari $Q^T A Q$ dengan A

adalah matriks awal. Kedua proses tersebut diulangi berkali-kali dan diagonal dari matriks hasil operasi yang terakhir adalah kumpulan nilai eigennya. Proses mendapatkan eigenvector juga berhubungan dengan proses mendapatkan *eigenvalue*. Vektor eigennya dapat diperoleh dengan mengali matriks semua Q yang didapat dalam proses iterasi.

2.7 Matriks Kovarian

Matriks kovarian adalah matriks yang mewakili nilai kovarian antara pasangan elemen dalam vektor acak. Kovarian sendiri berarti ukuran yang untuk melihat hubungan antara perubahan dalam satu variabel dengan perubahan dalam variabel lainnya. Matriks kovarian sendiri dapat dihitung menggunakan rumus sebagai berikut.

$$C = AA^T$$

2.8 Euclidean Distance

Secara umum, *euclidean distance* adalah perhitungan yang merepresentasikan jarak terpendek antara dua titik. Dalam konteks \mathbb{R}^n , euclidean distance di antara dua vektor adalah norma dari selisih kedua vektor tersebut. Misal, euclidean distance antara dua vektor $\mathbf{x} = (x_1, x_2, \dots, x_n)$ dan $\mathbf{y} = (y_1, y_2, \dots, y_n)$ adalah

$$ED(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$$

Euclidean distance merupakan salah satu *distance metric* yang banyak digunakan dalam bidang *machine learning*, terutama untuk menyelesaikan masalah *classification* dan *clustering*. Dengan *euclidean distance*, kemiripan antara dua hasil observasi yang direpresentasikan dalam vektor di ruang Euclidean dapat dihitung.

2.9 Image Normalization

Dalam konteks *image processing*, normalisasi adalah proses untuk mengubah range dari intensitas pixel yang ada pada gambar. Normalisasi akan mengubah sebuah vektor gambar *grayscale* $\mathbf{I}: \{\mathbb{X} \subseteq \mathbb{R}^n\}$ yang memiliki range intensitas pixel (Min,Max) menjadi sebuah *image* baru $\mathbf{I}_N: \{\mathbb{X} \subseteq \mathbb{R}^n\}$ dengan range intensitas pixel (newMin,newMax). Keduanya dihubungkan dengan persamaan

$$\mathbf{I}_N = \frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} (\mathbf{I} - \mathbf{Min}) + \mathbf{newMin} \quad (\text{pers. 2.8.1})$$

dengan \mathbf{Min} adalah vektor berdimensi n yang memiliki nilai $(\text{Min}, \text{Min}, \dots, \text{Min})$ dan \mathbf{newMin} adalah vektor berdimensi n yang memiliki nilai $(\text{newMin}, \text{newMin}, \dots, \text{newMin})$.

Metode normalisasi gambar ini dapat meningkatkan kualitas gambar yang memiliki kontras rendah. Metode ini digunakan dalam image preprocessing pada *face recognition*, untuk

menyamakan intensitas pixel maksimum dan minimum dari keseluruhan gambar yang digunakan.

BAB III

IMPLEMENTASI PROGRAM

3.1 Algoritma

Algoritma yang digunakan dalam pembuatan face recognition ini adalah algoritma eigenfaces (PCA *algorithm*) yang memanfaatkan konsep nilai dan vektor eigen pada matriks. Secara garis besar, algoritma ini terdiri dari tiga tahap, yaitu tahap *image preprocessing*, tahap *training* dan tahap *testing*.

3.1.1 Tahap *Image Preprocessing*

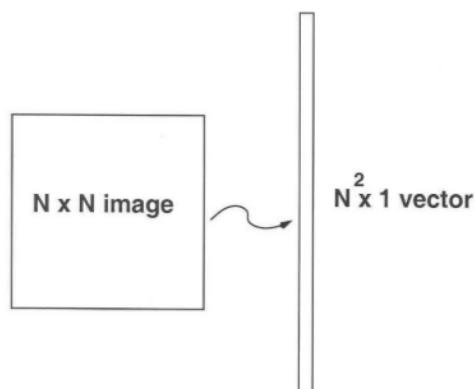
Tahap *image preprocessing* adalah tahap untuk menyamakan standar gambar yang akan dipakai. Tahap ini terdiri atas:

1. Mengubah gambar menjadi *grayscale* dengan ukuran 256×256 pixel
2. Menormalisasi gambar menjadi gambar dengan range intensitas pixel (0,255) sesuai persamaan 2.8.1

3.1.2 Tahap *Training*

Tahap *training* adalah tahap untuk “melatih mesin” dengan *dataset* (yang sudah di-*preprocessing*) yang dimiliki. Berikut adalah langkah-langkah pada tahap *training*.

1. Misalkan terdapat m buah gambar wajah (yang sudah di-*preprocessing*) pada *dataset*. Sesuai pembahasan sebelumnya, gambar ini akan memiliki dimensi 256×256 pixel. Ubah matriks setiap gambar menjadi vektor wajah berukuran 256^2 . Sebut vektor-vektor gambar ini $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$.



2. Hitung rata-rata ($\boldsymbol{\psi}$) dari seluruh vektor wajah ini.

$$\boldsymbol{\psi} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$$

3. Kurangkan setiap vektor wajah dengan vektor rata-rata. Untuk vektor wajah \mathbf{x}_i , akan dihasilkan vektor \mathbf{a}_i . Gabungkan vektor \mathbf{a}_i menjadi sebuah matriks A berukuran $256^2 \times m$.

$$\mathbf{a}_i = \mathbf{x}_i - \boldsymbol{\psi}$$

$$A = [\mathbf{a}_1 \ \mathbf{a}_2 \ \mathbf{a}_3 \ \dots \ \mathbf{a}_m]$$

4. Hitung matriks kovarian. Matriks kovarian (C) dapat dihitung dengan persamaan

$$C = AA^T$$

Dari matriks kovarian, nilai eigen (λ_i) dan vektor eigen (\mathbf{u}_i) akan dicari.

$$AA^T \mathbf{u}_i = \lambda_i \mathbf{u}_i$$

Namun, penggunaan persamaan di atas akan terlalu banyak memakan memori karena matriks yang dikalikan berukuran relatif sangat besar, dan matriks yang dihasilkan juga berukuran sangat besar $256^2 \times 256^2$. Oleh karena itu, dalam perhitungan kovarian, digunakan cara lain.

Untuk matriks A , nilai eigen dan vektor eigen dari $A^T A$ dinyatakan dengan persamaan

$$A^T A \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

Kalikan kedua ruas dengan matriks A .

$$\begin{aligned} AA^T A \mathbf{v}_i &= \lambda_i A \mathbf{v}_i \\ CA \mathbf{v}_i &= \lambda_i A \mathbf{v}_i \\ C \mathbf{u}_i &= \lambda_i \mathbf{u}_i \end{aligned}$$

Selanjutnya kita akan menyebut vektor ata sebagai *simplified covariance* (C'). Dari persamaan di atas dapat dilihat bahwa C' dan C akan memiliki nilai eigen yang sama, dan vektor eigen keduanya dihubungkan dengan persamaan

$$\mathbf{u}_i = A \mathbf{v}_i$$

Hal ini berarti kita cukup mencari nilai eigen dan vektor eigen dari matriks $A^T A$ yang memiliki dimensi relatif jauh lebih kecil dibanding AA^T , dan mengalikan vektor eigen dari $A^T A$ dengan matriks A . Vektor eigen dari kovarian (\mathbf{u}_i) akan dinormalisasi menjadi $\hat{\mathbf{u}}_i$. Vektor $\hat{\mathbf{u}}_i$ disebut *eigenface*. Vektor-vektor *eigenface* akan membangun *eigenspace* (ruang eigen).

5. Selanjutnya, pilih k nilai eigen terbesar (tersignifikan). Ambil eigenface yang bersesuaian dengan k nilai eigen tersebut.
6. Eigenface ini akan menjadi “basis” dari setiap wajah yang ada. Setiap vektor wajah akan dinyatakan dalam kombinasi linear *eigenface*

$$\mathbf{a}_i = \mathbf{x}_i - \boldsymbol{\psi} = \sum_{j=1}^k w_{ij} \hat{\mathbf{u}}_j$$

dengan w_{ij} adalah *weight* dari eigenface $\hat{\mathbf{u}}_j$ pada \mathbf{x}_i . Untuk mencari w_{ij} , digunakan persamaan berikut.

$$w_{ij} = \hat{\mathbf{u}}_j^T \mathbf{a}_i = \hat{\mathbf{u}}_j \cdot \mathbf{a}_i$$

7. Gabungkan koefisien *weight* dari gambar \mathbf{x}_i dalam vektor \mathbf{W}_i .

$$\mathbf{W}_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{ik} \end{bmatrix}$$

3.1.3 Tahap *Testing*

Tahap *testing* adalah tahap untuk mengetes gambar yang ingin dicek klasifikasinya. Berikut adalah langkah-langkahnya.

1. Ubah matriks gambar *testing* (yang sudah di-*preprocessing*) menjadi vektor berukuran 256^2 (\mathbf{x}_{test}).
2. Kurangkan vektor \mathbf{x}_{test} dengan vektor rata-rata ($\boldsymbol{\psi}$) yang sudah dihitung pada tahap training untuk menghasilkan vektor \mathbf{a}_{test} .

$$\mathbf{a}_{test} = \mathbf{x}_{test} - \boldsymbol{\psi}$$

3. Training image akan dinyatakan dalam kombinasi linear *eigenface*

$$\mathbf{a}_{test} = \mathbf{x}_{test} - \boldsymbol{\psi} = \sum_{j=1}^k w_{test,j} \hat{\mathbf{u}}_j$$

dengan $w_{test,j}$ adalah *weight* dari eigenface $\hat{\mathbf{u}}_j$ pada \mathbf{x}_{test} . Untuk mencari $w_{test,j}$, digunakan persamaan berikut.

$$w_{test,j} = \hat{\mathbf{u}}_j^T \mathbf{a}_{test} = \hat{\mathbf{u}}_j \cdot \mathbf{a}_{test}$$

4. Gabungkan koefisien *weight* dari gambar \mathbf{x}_{test} dalam vektor \mathbf{W}_{test} .

$$\mathbf{W}_{test} = \begin{bmatrix} w_{test,1} \\ w_{test,2} \\ \vdots \\ w_{test,k} \end{bmatrix}$$

5. Cari vektor \mathbf{W}_i yang memiliki *euclidean distance* terkecil (ED_{min}) dengan \mathbf{W}_{test} . Gambar yang bersesuaian dengan vektor \mathbf{W}_i berarti adalah gambar terdekat dengan gambar *testing*.
6. Suatu gambar dikatakan terklasifikasi ke dalam *dataset* jika ED_{min} memenuhi ketidaksamaan berikut.

$$ED_{min} < T$$

T adalah *threshold* yang biasanya dihitung dengan persamaan

$$T = \frac{1}{2}ED_{max}$$

3.2 Tech Stack

Seluruh bagian dari program *Face Recognition with Eigenface* diimplementasikan dalam bahasa pemrograman Python. Python adalah bahasa pemrograman *high-level* dan bisa digunakan untuk berbagai tujuan. Bahasa pemrograman Python juga memiliki *package manager* (pip) yang mudah untuk digunakan dan memiliki berbagai *libraries* untuk berbagai tujuan.

Pada program ini digunakan berbagai *library* Python, yaitu:

1. OpenCV

OpenCV adalah *library computer vision* Python. *Library* ini memungkinkan kita untuk melakukan pemrosesan gambar, mulai dari pengambilan gambar, mengubah ukuran gambar, mengubah warna gambar, dan melakukan transformasi gambar menjadi matriks.

2. Numpy

Numpy adalah *library* Python untuk memudahkan proses komputasi dan perhitungan matriks dan vektor. Numpy digunakan untuk beberapa proses seperti perkalian matriks, transpose matriks, dan sebagainya.

3. Tkinter

Tkinter adalah *library* Python untuk mengimplementasi *Graphical User Interface* (GUI) program. Pemilihan library ini didasari atas penggunaanya yang *beginner-friendly* dan bisa mencukupi fitur yang diperlukan oleh program ini.

4. Pillow (PIL)

Pillow adalah *library* Python untuk menampilkan image pada GUI. Image yang akan ditampilkan adalah *image* untuk *test image* dan hasil *recognition* program.

5. Timeit/Time

Timeit/Time adalah *library* Python yang digunakan untuk menghitung waktu eksekusi program sehingga dapat ditampilkan pada GUI.

6. OS

OS adalah *library* Python yang menyediakan fungsi untuk berinteraksi dengan sistem operasi. *Library* ini digunakan untuk mengurus segala keperluan *file* dan *folder*, misalnya untuk mendapatkan *path* dari suatu *folder* ataupun *file*.

3.3 Program

3.3.1 Modul dan File

Program kami terdiri dari beberapa modul dan file sebagai berikut.

No.	Modul	File	Isi
1	configImages	configImages.py	konfigurasi gambar, <i>preprocessing</i> gambar
2	function	function.py	pengolahan vektor dan matriks, pencarian nilai eigen dan vektor eigen, <i>face recognition</i>
3	gui	gui.py	pengaturan tampilan (<i>interface</i>) aplikasi dan sebagai <i>main program</i>

3.3.2 Modul configImages

No.	Fungsi	Keterangan
1	readImage(pathFile)	Mengubah suatu gambar dengan <i>path</i> <i>pathFile</i> menjadi matriks berwarna <i>grayscale</i> dan berukuran 256×256
2	convertFrame(frame)	Mengubah suatu matriks <i>frame</i> menjadi <i>grayscale</i> , dan berukuran 256×256
3	resizeImage(source)	Me- <i>resize</i> dan mengonversikan <i>array NumPy</i> menjadi PIL Image
4	filesInsideFolder(folderName, listFiles)	Mengembalikan <i>list</i> berisi <i>path-path</i> dari setiap <i>file images</i> yang ada di folder yang bernama <i>folderName</i> . Fungsi ini mengabaikan file dengan format selain format gambar yang disupport oleh <i>library OpenCV</i> .

3.3.3 Modul function

No.	Fungsi	Keterangan
1	normalizeImg(imgVec)	Menormalisasikan gambar menjadi gambar dengan range intensitas pixel (0,255)
2	normalize(vec)	Menormalisasi suatu vektor

3	<code>euclidean_norm(vector)</code>	Menghitung panjang vektor
4	<code>orthogonal_matrix(matrix)</code>	Menemukan matriks ortogonal (Q) dari dekomposisi QR menggunakan prosedur Gram-Schmidt
5	<code>upper_triangle(matrix, Q)</code>	Menemukan matriks segitiga atas (R) dari dekomposisi QR menggunakan prosedur Gram-Schmidt menggunakan Q yang didapatkan dari dekomposisi QR menggunakan prosedur Gram-Schmidt
6	<code>eigen(matrix)</code>	Menemukan nilai eigen atau <i>eigenvalue</i> dan vektor eigen atau <i>eigenvector</i> dari suatu matriks persegi yang simetri
7	<code>testImgCam(rawTestImgMat)</code>	<i>Preprocessing</i> untuk gambar yang diterima dari kamera
8	<code>testImgFile(pathfile)</code>	<i>Preprocessing</i> untuk gambar yang diterima dari pathFile
9	<code>faceRecog(pathfolder, testImgMat)</code>	Implementasi keseluruhan <i>face recognition</i> , dari tahap <i>preprocessing</i> hingga <i>testing</i>

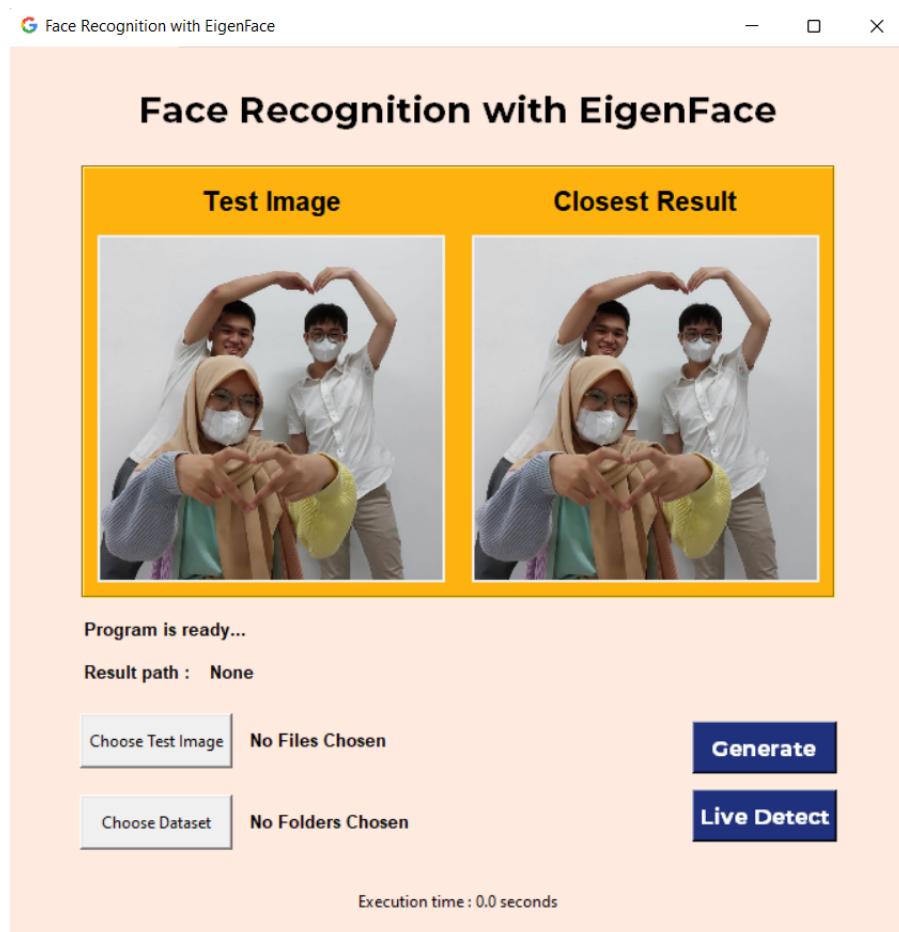
3.3.4 GUI dan Program Utama

Program utama *Face Recognition with Eigenface* diimplementasikan pada file `gui.py`. File `gui.py` adalah file yang berfungsi sebagai main program sekaligus GUI program. Pada GUI, user bisa memilih file test image dan folder dataset. Pengambilan test image juga bisa dilakukan secara *live* melalui kamera perangkat dengan menekan tombol *Live Detect*, lalu program akan menunggu selama 5 detik sebelum mengambil foto. Program akan menampilkan foto *test image*, *path* dan foto hasil rekognisi yang disertai dengan persentase kedekatan antara dua foto tersebut. Tombol *Generate* berfungsi untuk memulai proses rekognisi dan waktu yang digunakan untuk melakukan rekognisi akan ditampilkan pada bagian bawah program.

Program utama menggunakan modul-modul yang telah dibuat seperti modul `configImages` dan `function`. Modul bawaan Python yang digunakan adalah `tkinter`, `PIL`, `cv2`, dan `time`. Terdapat beberapa modul yang dibuat pada GUI:

No.	Fungsi	Keterangan
-----	--------	------------

1	chooseTest()	Memilih file yang akan digunakan sebagai testImage
2	chooseDataset()	Memilih folder yang akan digunakan sebagai dataset
3	generate()	Melakukan proses rekognisi pada program
4	detect()	Melakukan pengambilan test image secara <i>live</i> dari kamera perangkat

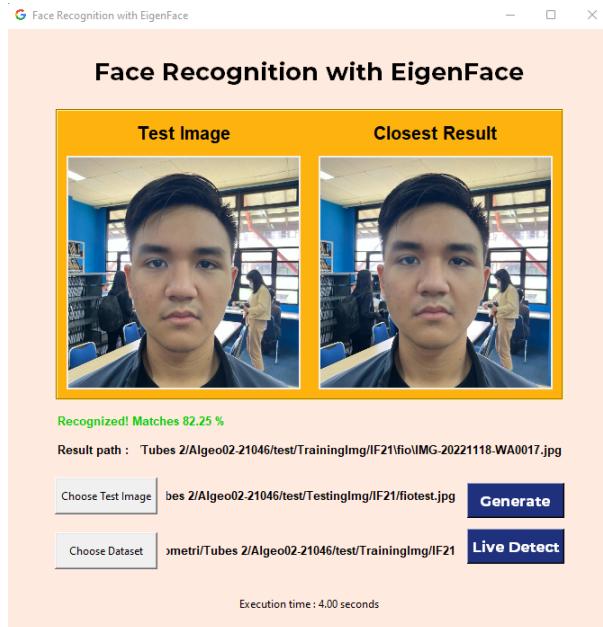


Gambar 3.3.4 Tampilan program pada GUI

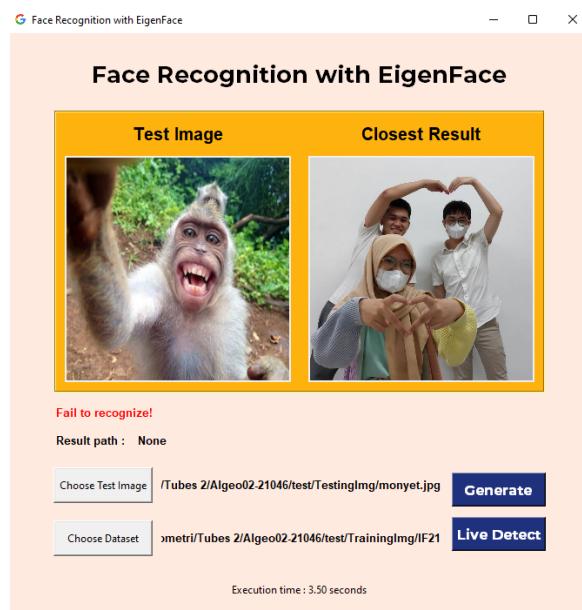
BAB IV

EKSPERIMEN

4.1 Masukan *Testing Image* dari File

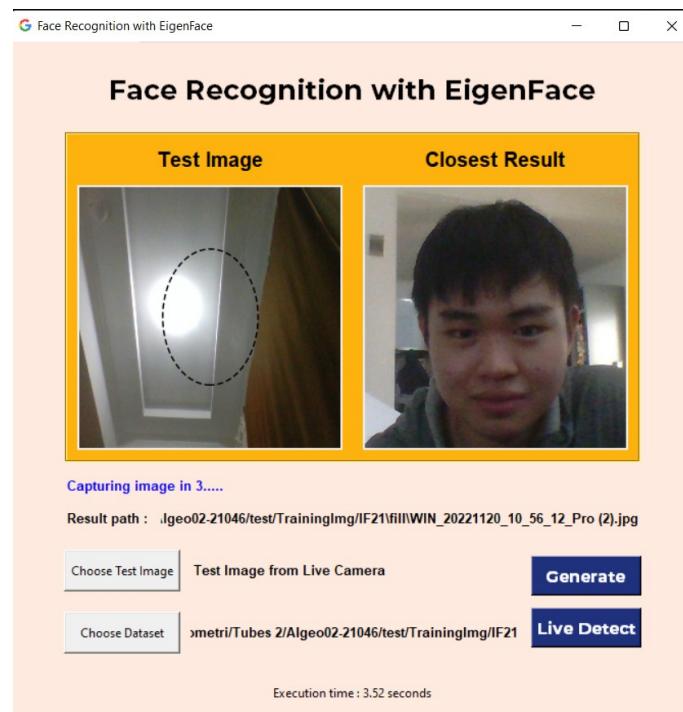


Gambar 4.1.1 Tampilan ketika masukan *testing image* dari file berhasil dikenali (*recognized*) dan hasilnya benar

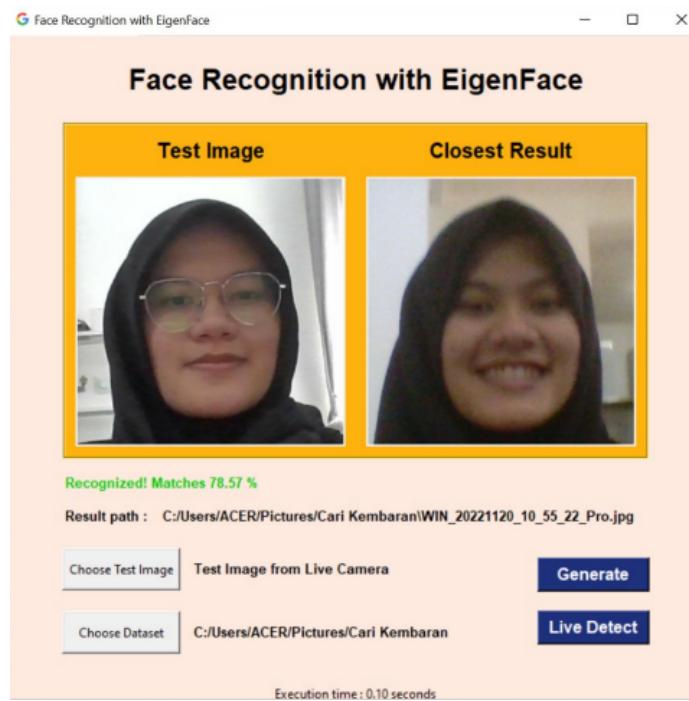


Gambar 4.1.2 Tampilan ketika masukan *testing image* dari file tidak berhasil dikenali (*fail to recognize*)

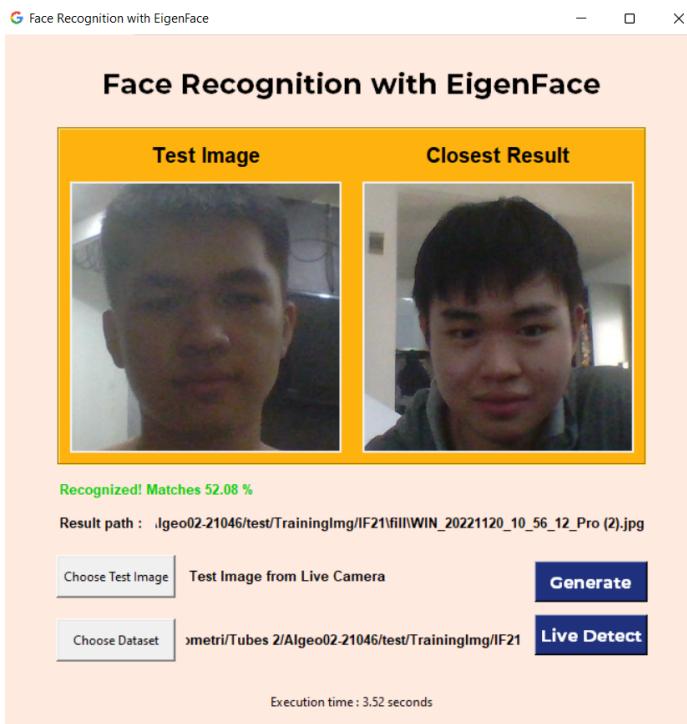
4.2 Masukan *Testing Image* dari Kamera



Gambar 4.2.1 Tampilan aplikasi saat mengambil masukan dari kamera



Gambar 4.2.2 Tampilan ketika masukan *testing image* dari kamera berhasil dikenali (*recognized*) dan hasilnya benar



Gambar 4.2.3 Tampilan ketika masukan *testing image* dari kamera berhasil dikenali (*recognized*) namun hasilnya salah

4.3 Analisis

Pada hasil eksperimen di atas, dapat dilihat bahwa:

- Aplikasi sudah mampu mengenali gambar yang ada di dataset dan menghasilkan *closest image result* yang sesuai dengan masukan gambar yang dipilih dari *file* (contohnya Gambar 4.1.1) atau diambil dari kamera (contohnya Gambar 4.2.2).
- Aplikasi sudah mampu membedakan mana gambar yang dikenali (contohnya Gambar 4.1.1) dan gambar yang tidak dikenali (contohnya Gambar 4.1.2). Pada kasus Gambar 4.1.2, gambar yang dimasukkan sebagai *testing image* adalah gambar kera yang jelas tidak ada pada dataset wajah pada *folder* yang dipilih.
- Aplikasi bisa salah mengenali wajah (contohnya Gambar 4.2.3). Pada kasus Gambar 4.2.3, gambar dikenali, namun hasil *closest image result*-nya tidak sesuai. Hal ini bisa terjadi karena banyak faktor. Seperti yang dapat dilihat pada Gambar 4.2.3, gambar masukan *testing image* memiliki kualitas *lighting* yang kurang bagus dan wajah yang berada pada *testing image* memang lumayan mirip dengan gambar yang ada pada *closest image result*.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

5.1 Kesimpulan

Dari hasil tugas besar II IF2123 Aljabar Linier dan Geometri ini, kami berhasil memperoleh kesimpulan-kesimpulan sebagai berikut.

- *Face Recognition* atau pengenalan wajah merupakan teknologi biometrik untuk mengidentifikasi wajah manusia yang dapat digunakan untuk berbagai kepentingan, khususnya di bidang keamanan.
- Ada berbagai metode yang bisa digunakan untuk melakukan pencocokan suatu wajah dengan wajah lain dalam suatu kumpulan citra wajah, salah satunya adalah *Eigenface*.
- Metode *Eigenface* melibatkan perhitungan *eigenvalue* atau nilai eigen dan *eigenvector* atau vektor eigen.
- Metode *Eigenface* berhasil kami gunakan untuk mengimplementasikan *Face Recognition* ini.

5.2 Saran

Dari pelaksanaan tugas besar II IF2123 Aljabar Linier dan Geometri ini, kami menyarankan untuk mengadakan fitur untuk pembersihan *dataset* serta memakai *image preprocessing* yang lebih baik lagi.

5.3 Refleksi

Dari hasil tugas besar II IF2123 Aljabar Linier dan Geometri ini, kami mendapatkan refleksi-refleksi sebagai berikut.

- Kami berhasil mendapatkan *eigenvalue* dan *eigenvector* tanpa menggunakan fungsi eigen yang ada di library bahasa pemrograman Python.
- Kami berhasil mengimplementasikan metode *Eigenface* untuk merealisasikan *Face Recognition*.
- Kami berhasil membuat GUI (*Graphical User Interface*) yang menarik untuk mendukung *Face Recognition* yang kami buat untuk tugas besar II IF2123 Aljabar Linier dan Geometri ini.

REFERENSI

Munir, Rinaldi. 2022. IF2123 Aljabar Geometri - Semester I Tahun 2022/2023.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2022-2023/algeo22-23.htm>

Pawangfg. 2021. *ML - Face Recognition Using Eigenfaces (PCA Algorithm)*.

<https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/>

Yanovsky, Igor. 2007. *QR Decomposition with Gram-Schmidt*.

<https://www.math.ucla.edu/~yanovsky/Teaching/Math151B/handouts/GramSchmidt.pdf>

LAMPIRAN

Repository GitHub: <https://github.com/ashnchiquita/Algeo02-21046.git>

Video YouTube: <https://youtu.be/LLFpKsNfvZw>