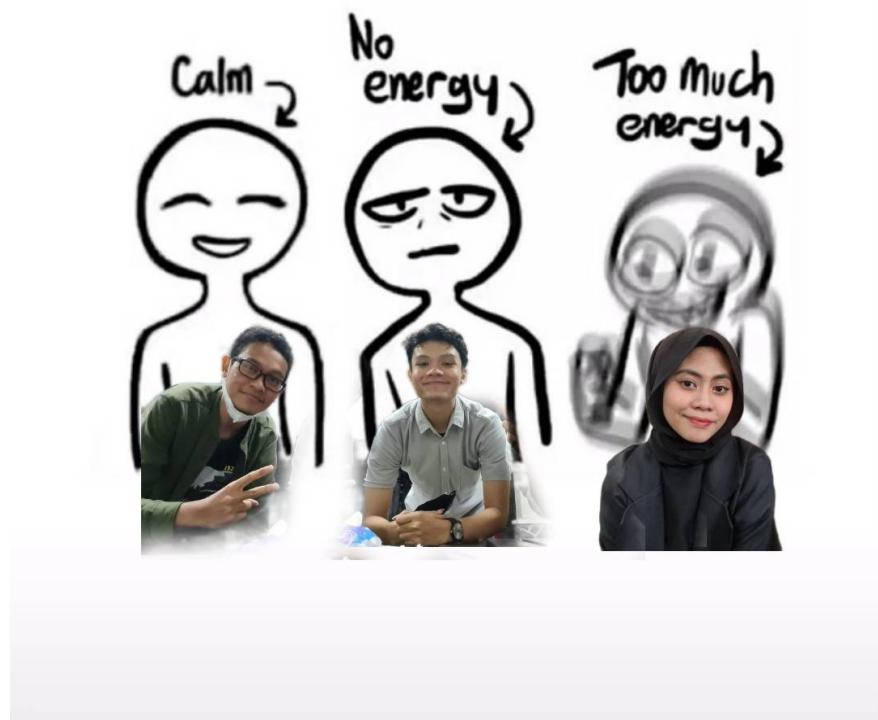


Tugas Besar 2 IF 2123

Aljabar Linier dan Geometri Aplikasi Nilai Eigen dan EigenFace pada Pengenalan Wajah (Face Recognition)



Oleh:

Muhammad Rifko Favian

13521075

Febryan Arota Hia

13521120

Dhanika Novlisariyanti

13521132

DAFTAR ISI

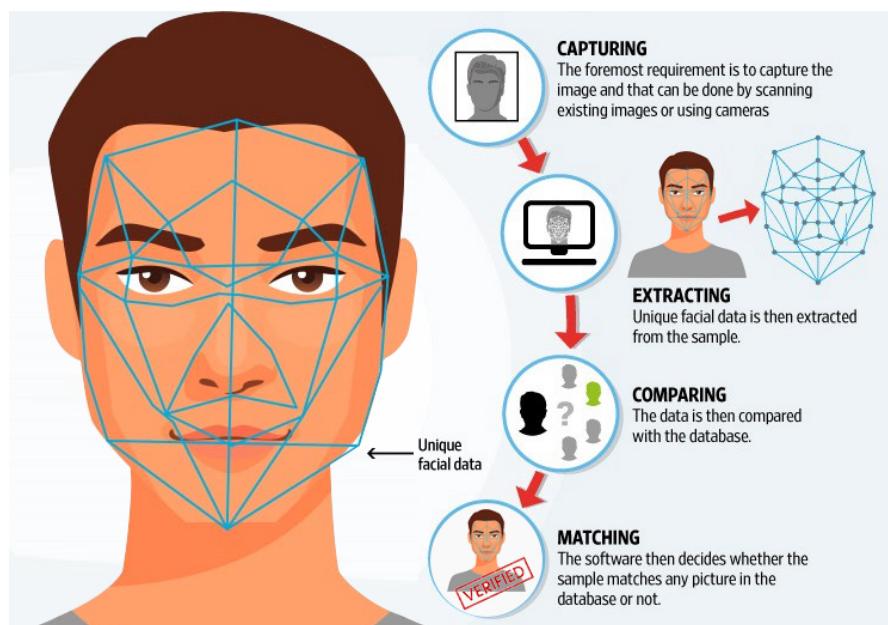
BAB I	3
DESKRIPSI MASALAH	3
1.1 Abstraksi	3
BAB II	5
LANDASAN TEORI	5
2.1 Perkalian Matriks	5
2.2 Nilai Eigen dan Vektor Eigen	5
Gambar 2.3 Vektor eigen	6
Gambar 2.4 Contoh perhitungan vektor eigen	6
2.3 Eigenface	7
BAB III	9
IMPLEMENTASI PROGRAM	9
3.1 Tech Stack yang digunakan	9
3.2 Algoritma Pencarian Eigen Value dan Eigen Vector	9
3.3 Algoritma Pengenalan Wajah	10
3.4 Implementasi Fungsi	11
3.4.1 Extract.py	11
3.4.2 Eigen.py	11
3.4.3 faceDetection.py	11
3.4.4 app.py	12
BAB IV	13
EKSPERIMEN	13
4.1 Eksperimen I	13
4.2 Eksperimen II	15
4.3 Eksperimen III	19
4.4 Eksperimen IV	22
4.5 Keterangan	25
BAB V	26
PENUTUP	26
DAFTAR REFERENSI	27
LAMPIRAN	28

BAB I

DESKRIPSI MASALAH

1.1 Abstraksi

Pengenalan wajah (*Face Recognition*) adalah teknologi biometrik yang bisa dipakai untuk mengidentifikasi wajah seseorang untuk berbagai kepentingan khususnya keamanan. Program pengenalan wajah melibatkan kumpulan citra wajah yang sudah disimpan pada database lalu berdasarkan kumpulan citra wajah tersebut, program dapat mempelajari bentuk wajah lalu mencocokkan antara kumpulan citra wajah yang sudah dipelajari dengan citra yang akan diidentifikasi. Alur proses sebuah sistem pengenalan wajah diperlihatkan pada Gambar 1.



Gambar 1.1 Alur proses di dalam sistem pengenalan wajah (Sumber: <https://www.shadowsystem.com/page/20>)

Terdapat berbagai teknik untuk memeriksa citra wajah dari kumpulan citra yang sudah diketahui seperti jarak Euclidean dan *cosine similarity*, principal component analysis (PCA), serta Eigenface. Pada Tugas ini, akan dibuat sebuah program pengenalan wajah menggunakan Eigenface.

Sekumpulan citra wajah akan digunakan dengan representasi matriks. Dari representasi matriks tersebut akan dihitung sebuah matriks Eigenface. Program pengenalan wajah dapat dibagi menjadi 2 tahap berbeda yaitu tahap *training* dan pencocokan. Pada tahap *training*, akan diberikan kumpulan data set berupa citra wajah. Citra wajah tersebut akan dinormalisasi dari RGB ke Grayscale (matriks), hasil normalisasi akan digunakan dalam perhitungan eigenface. Seperti namanya, matriks eigenface menggunakan eigenvector dalam pembentukannya. Berikut merupakan langkah rinci dalam pembentukan eigenface.

Program dibuat dengan Bahasa Python dengan memanfaatkan sejumlah library di OpenCV (Computer Vision) atau library pemrosesan gambar lainnya (contoh PIL). Fungsi untuk mengekstraksi fitur dari sebuah citra wajah tidak perlu anda buat lagi, tetapi menggunakan fungsi ekstraksi yang sudah tersedia di dalam library. Fungsi Eigen dilarang import dari library dan harus diimplementasikan, sedangkan untuk operasi matriks lainnya silahkan menggunakan library.

BAB II

LANDASAN TEORI

2.1 Perkalian Matriks

Perkalian antara dua matriks misalnya matriks A dan B , hanya bisa dilakukan jika jumlah kolom A sama dengan jumlah baris B . Perkalian tersebut akan menghasilkan suatu matriks dengan jumlah baris sama dengan baris matriks A dan jumlah kolom sama dengan kolom matriks B .

Jika A adalah matriks berukuran $m \times n$ dan B adalah matriks berukuran $n \times p$, dengan elemen-elemen sebagai berikut

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

Gambar 2.1 Contoh perkalian matriks

Hasil perkalian kedua matriks tersebut, $C = AB$ adalah sebuah matriks berukuran $m \times p$

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Gambar 2.2 Contoh hasil perkalian matriks

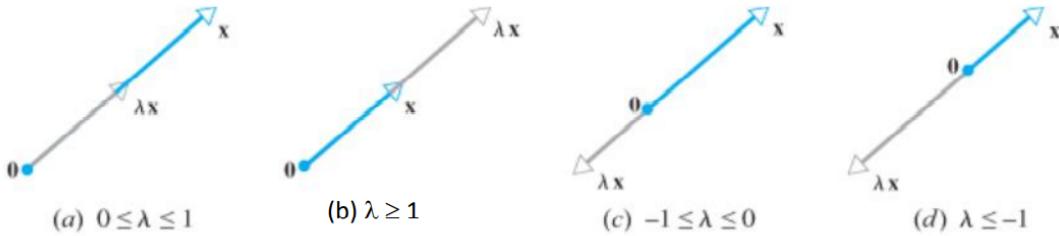
2.2 Nilai Eigen dan Vektor Eigen

Jika A adalah matriks $n \times n$ maka vektor tidak-nol x di R^n disebut vektor eigen dari A jika Ax sama dengan perkalian suatu skalar λ dengan x , yaitu

$$Ax = \lambda x$$

Skalar λ disebut nilai eigen dari A , dan x dinamakan vektor eigen yang berkoresponden dengan λ . Dengan kata lain, nilai eigen menyatakan nilai karakteristik dari sebuah matriks yang berukuran $n \times n$.

Vektor eigen x menyatakan matriks kolom yang apabila dikalikan dengan sebuah matriks $n \times n$ menghasilkan vektor lain yang merupakan kelipatan vektor itu sendiri. Dengan kata lain, operasi $Ax = x$ menyebabkan vektor x menyusut atau memanjang dengan faktor λ dengan arah yang sama jika λ positif dan arah berkebalikan jika λ negatif.



Gambar 2.3 Vektor eigen

Cara mendapatkan vektor eigen dan nilai eigen dari sebuah matriks A berukuran $n \times n$, dapat dihitung sebagai berikut.

$$\begin{aligned} Ax &= \lambda x \\ IAx &= \lambda Ix \\ Ax &= \lambda Ix \\ (\lambda I - A)x &= 0 \end{aligned}$$

$x = 0$ adalah solusi trivial dari $(\lambda I - A)x = 0$

Agar $(\lambda I - A)x = 0$ memiliki solusi tidak-nol, maka haruslah $\det(\lambda I - A)x = 0$

Persamaan $\det(\lambda I - A)x = 0$ disebut persamaan karakteristik dari matriks A , dan akar-akar persamaan tersebut, yaitu λ , dinamakan akar-akar karakteristik atau nilai-nilai eigen.

$$(\lambda I - A)x = 0 \rightarrow \begin{bmatrix} \lambda & -3 & 0 \\ -8 & \lambda + 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\text{Untuk } \lambda = 3 \rightarrow \begin{bmatrix} 0 & 0 \\ -8 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow -8x_1 + 4x_2 = 0 \rightarrow 8x_1 = 4x_2 \rightarrow x_1 = \frac{1}{2}x_2$$

\rightarrow Solusi: $x_1 = \frac{1}{2}t, x_2 = t, t \in \mathbb{R}$

$$\text{Vektor eigen: } x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}t \\ t \end{bmatrix} = t \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} \rightarrow \text{membentuk ruang eigen (eigenspace)}$$

Gambar 2.4 Contoh perhitungan vektor eigen

Selain cara di atas, terdapat juga algoritma lain untuk menghitung nilai eigen dan vektor eigen, seperti Lanczos Algorithm, Power Iteration, Inverse Iteration, Rayleigh Quotient Iteration, dan QR Algorithm.

2.3 Eigenface

Eigenface merupakan suatu algoritma yang menggunakan Principal Component Analysis (PCA) untuk mengurangi dimensionalitas serta untuk mencari vektor terbaik guna mendistribusikan citra wajah ke dalam ruang wajah yang ada. Tujuan utama dari PCA adalah untuk mencari vektor yang paling cocok yang dapat menggambarkan distribusi citra wajah di dalam ruang citra ke dalam ruang wajah.

Eigenvector dan eigenvalue didapatkan dari matriks kovarian yang degenerate dari citra wajah yang dilatih. Eigenvector diurutkan berdasarkan eigenvalue-nya dan dipilih sebanyak m eigenvector pertama untuk membentuk principal component. Eigenvector diturunkan dari sebuah matriks kovarian dari ruang vektor yang mungkin dari wajah-wajah manusia.

Berikut langkah umum algoritma Eigenface:

1. Langkah pertama adalah menyiapkan data dengan membuat suatu himpunan S yang terdiri dari seluruh training image, $(\Gamma_1, \Gamma_2, \dots, \Gamma_M)$

$$S = (\Gamma_1, \Gamma_2, \dots, \Gamma_M) \quad (1)$$

2. Langkah kedua adalah ambil nilai rata-rata atau mean (Ψ)

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n \quad (2)$$

3. Langkah ketiga kemudian cari selisih (ϕ_i) antara nilai training image (Γ_i) dengan nilai tengah (Ψ)

$$\phi_i = \Gamma_i - \Psi \quad (3)$$

4. Langkah keempat adalah menghitung nilai matriks kovarian (C)

$$C = \frac{1}{M} \sum_{n=1}^M \phi_n \phi_n^T = A A^T$$
$$L = A^T A \quad L = \phi_m^T \phi_m \quad (4)$$

5. Langkah kelima menghitung eigenvalue (λ) dan eigenvector (v) dari matriks kovarian (C)

$$C \times v_i = \lambda_i \times v_i \quad (5)$$

6. Langkah keenam, setelah eigenvector (v) diperoleh, maka eigenface (μ) dapat dicari dengan:

$$\mu_i = \sum_{k=1}^M v_{ik} \phi_k \quad (6)$$

Tahapan Pengenalan wajah :

1. Sebuah image wajah baru atau test face (Γ_{new}) akan dicoba untuk dikenali, pertama terapkan cara pada tahapan pertama perhitungan eigenface untuk mendapatkan nilai eigen dari image tersebut.

$$\mu_{new} = v \times \Gamma_{new} - \Psi \quad (7)$$

$$\Omega = \mu_1, \mu_2, \dots, \mu_M$$

2. Gunakan metode euclidean distance untuk mencari jarak (distance) terpendek antara nilai eigen dari training image dalam database dengan nilai eigen dari image testface. $\varepsilon_k = \Omega - \Omega_k$ (8)

Pada tahapan akhir, akan ditemui gambar dengan euclidean distance paling kecil maka gambar tersebut yang dikenali oleh program paling menyerupai test face selama nilai kemiripan di bawah suatu nilai batas. Jika nilai minimum di atas nilai batas maka dapat dikatakan tidak terdapat citra wajah yang mirip dengan test face.

BAB III

IMPLEMENTASI PROGRAM

3.1 Tech Stack yang digunakan

GUI yang digunakan dalam program ini adalah menggunakan Tkinter. Tkinter adalah *interface* yang biasa digunakan bersamaan dengan bahasa pemrograman Python. Selain itu, digunakan juga library *OpenCV* untuk meng-extract foto dengan metode KAZE. Selain itu, digunakan juga library PIL untuk mengolah foto. Untuk mengolah hasil extract matriks, penulis menggunakan library numpy dan sympy.

3.2 Algoritma Pencarian Eigen Value dan Eigen Vector

Pada pencarian nilai eigen value dan eigen vector dalam program ini, kami menggunakan algoritma *power iteration*, sebagai berikut:

- a. Misal A adalah suatu matriks simetris, dekomposisi eigen dari A adalah $A = Q \wedge QT$, apabila qi merupakan kolom dari Q , adalah nilai eigen dominan apabila $\lambda_1 > \lambda_i$ untuk semua $i = 2, 3, 4, \dots$ dan eigen vektor yang berkoresponden juga berarti merupakan eigen vektor dominan, dominan disini maksudnya dengan nilai terbesar.
- b. Untuk melakukan hal ini, digunakan Power Method yaitu pertama-tama, diambil unit vector random x_0 , sehingga $x_1 = A \cdot x_0$, $x_2 = A \cdot A \cdot x_0$, dan seterusnya sampai dia mendekati nilai sebenarnya.

Metode diatas hanya menemukan vektor eigen paling besar, namun karena kita mengetahui bahwa vektor eigen lain ortogonal kepada eigen vektor dominan, bisa digunakan lagi power method untuk memaksa vektor kedua yang dihasilkan merupakan suatu vektor yang ortogonal dari vektor pertama, hal ini dilakukan terus menerus sampai ditemukan semua vektor eigen dari suatu matriks. Untuk algoritma yang digunakan pada kode ini:

- a. Ambil Q sehingga $QT \cdot QT = I$, lalu dilakukan iterasi berupa 00
- b. $Zk = A \cdot Qk-1 \cdot Zk-1$
- c. $Qk \cdot Rk = Zk$ (dalam hal ini digunakan dekomposisi QR)

Metode dekomposisi QR yang kami implementasi menggunakan metode *Gram-Schmidt Orthogonalization*. Metode ini banyak digunakan untuk mencari matriks ortogonal Q dari A secara rekursif, berdasarkan proyeksi ortogonal dari setiap vektor $a \in A$, $k = 1..n$ ke dalam rentang vektor $q_i \in Q(k)$, $i = 1..k$. Setiap vektor ortogonal baru $q \in Q$ dihitung sebagai jumlah proyeksi, mengurangkannya dari vektor yang sesuai $a \in A$. Akhirnya, matriks

segitiga atas R diperoleh sebagai perkalian transpos Q dan matriks A , dengan menggunakan persamaan (1) berikut.

$$R = Q^T A \quad (1.1)$$

$$Q^T Q = Q Q^T = I \quad (1.2)$$

Proyeksi ortogonal \vec{a} ke \vec{q} diperoleh dengan menggunakan persamaan (2) berikut:

$$\text{proj}_{\vec{q}} \vec{a} = \frac{\langle \vec{q}, \vec{a} \rangle}{\langle \vec{q}, \vec{q} \rangle} * \vec{q} = \frac{\langle \vec{q}, \vec{a} \rangle}{\|\vec{q}\|^2} * \vec{q} \quad (2)$$

Biasanya, seluruh proses Gram-Schmidt menghasilkan matriks ortogonal Q dan dapat dinyatakan sebagai:

$$\vec{q}_k = \vec{a}_k - \sum_{i=1}^k \text{proj}_{\vec{q}_i} \vec{a}_k, \quad (3.1)$$

$$\vec{q}_k = \frac{\vec{q}_k}{\|\vec{q}_k\|} \quad (3.2)$$

, atau alternatifnya:

$$\vec{q}_1 = \vec{a}_1,$$

$$\vec{q}_2 = \vec{a}_2 - \text{proj}_{\vec{q}_1} \vec{a}_2,$$

$$\vec{q}_3 = \vec{a}_3 - \text{proj}_{\vec{q}_1} \vec{a}_3 - \text{proj}_{\vec{q}_2} \vec{a}_3, |$$

$$\vec{q}_4 = \vec{a}_4 - \text{proj}_{\vec{q}_1} \vec{a}_4 - \text{proj}_{\vec{q}_2} \vec{a}_4 - \text{proj}_{\vec{q}_3} \vec{a}_4$$

...

Setiap vektor baru $\vec{q} \in Q$ akhirnya dinormalisasi dengan membaginya dengan norm Euclidean-nya $|\vec{q}|$.

3.3 Algoritma Pengenalan Wajah

Pada ekstraksi wajah ini, penulis menggunakan algoritma KAZE. Algoritma KAZE adalah fitur deteksi yang beroperasi di skala ruang nonlinear. Dengan metode KAZE, foto di ekstrak dengan menyimpan informasi penting detail foto dan menghilangkan *noise* sekitar foto sampai foto menjadi skala ruang nonlinear.

3.4 Implementasi Fungsi

3.4.1 Extract.py

Nama Fungsi	Keterangan
extract_features	Fungsi ini menghasilkan matriks hasil ekstraksi tiap image
batch_extractor	Fungsi ini mendapatkan input folder dan dipisahkan menjadi array nama dan isi matriks hasil ekstraksi melalui extract_features

3.4.2 Eigen.py

Nama Fungsi	Keterangan
mean	Fungsi ini digunakan untuk mencari rata-rata matriks
selisih	Fungsi ini digunakan untuk mengurangi suatu matriks dengan suatu nilai (dalam hal ini adalah mean)
covarian	Fungsi ini menghasilkan matriks covarian
eigen_qr	Fungsi ini digunakan untuk QR decomposition
make_householder	Fungsi ini menghasilkan vektor normalisasi yang digunakan dalam fungsi eigen_qr
qr_iteration	Fungsi ini menghasilkan eigen value dan eigen vector dengan power iteration method
eigenFace	Fungsi ini menghasilkan nilai eigenFace
weightFace	Fungsi ini menghasilkan weightFace

3.4.3 faceDetection.py

Nama Fungsi	Keterangan
extractFace	Fungsi ini digunakan untuk ekstraksi <i>image</i> yang akan diuji

queryWeight	Fungsi ini digunakan untuk mencari nilai <i>weight face image</i> yang diuji
euclideanDistance	Fungsi ini digunakan untuk mencari nilai <i>euclidean distance</i>
bestMatch	Fungsi ini mengembalikan nilai <i>euclidean distance</i> terkecil

3.4.4 app.py

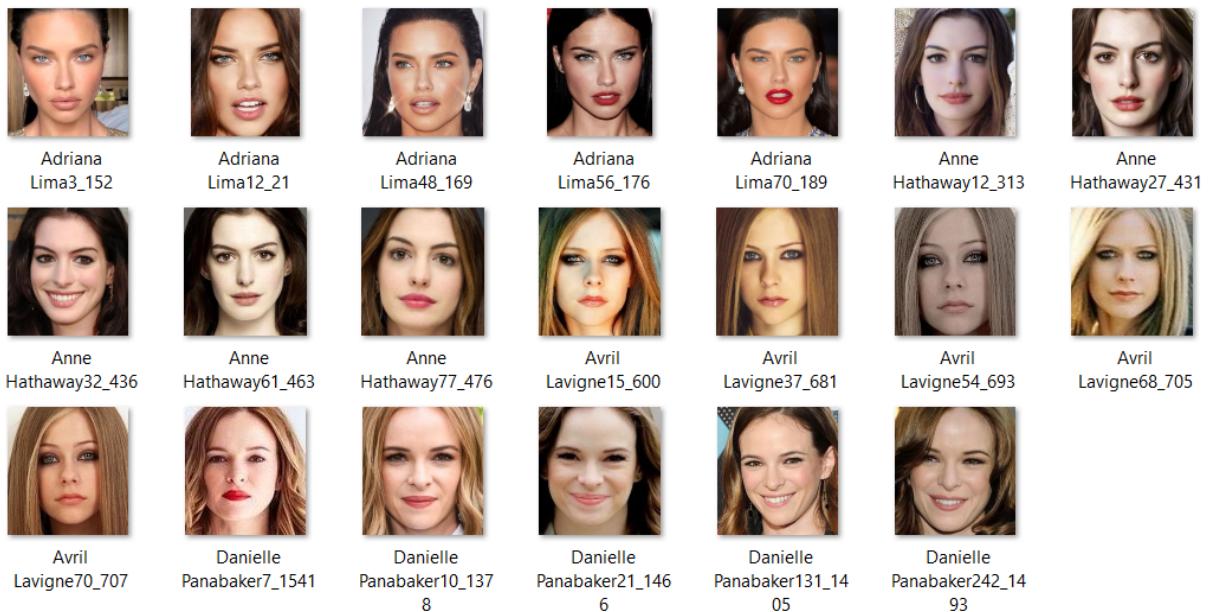
Nama Fungsi	Keterangan
combineFunc	Fungsi ini berfungsi untuk menggabungkan <i>command command</i> dalam button
openData	Fungsi ini berfungsi untuk membuka data dan memproses data folder tersebut
openImage	Fungsi ini berfungsi untuk membuka image yang akan di extract dan memproses <i>image</i> tersebut agar bisa di deteksi
openCamera	Fungsi ini berfungsi untuk membuka kamera dan mengcapture foto dalam kurun waktu 7 detik
run	Fungsi ini berfungsi untuk menjalankan aplikasi Tkinter

BAB IV

EKSPERIMENT

4.1 Eksperimen I

Pada eksperimen I, digunakan dataset 4 orang berbeda masing-masing 5 foto dengan total 20 foto.



Gambar 4.1.1 Dataset 1

FACE RECOGNITION

Manual

Input Your Dataset

Choose Folder
Sucsesfull choosed

Input Your Image

Choose File
Anne Hathaway34_438.jpg

Open Camera

Result

Anne Hathaway27_431.jpg

Test Image

Closest Result

Execution time: 2.8038511276245117

Gambar 4.1.2

FACE RECOGNITION

Manual

Input Your Dataset

Choose Folder

Succesfull choosed

Input Your Image

Choose File

Avril Lavigne12_579.jpg

Open Camera

Result

Avril Lavigne37_681.jpg



Gambar 4.1.3

FACE RECOGNITION

Manual

Input Your Dataset

Choose Folder

Succesfull choosed

Input Your Image

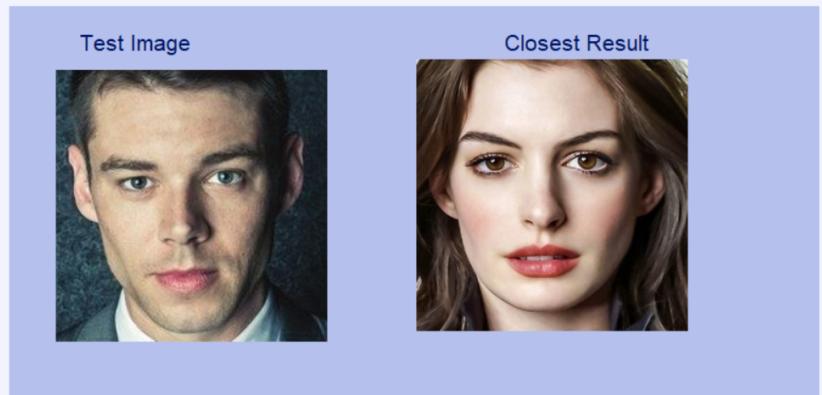
Choose File

Brian J. Smith14_630.jpg

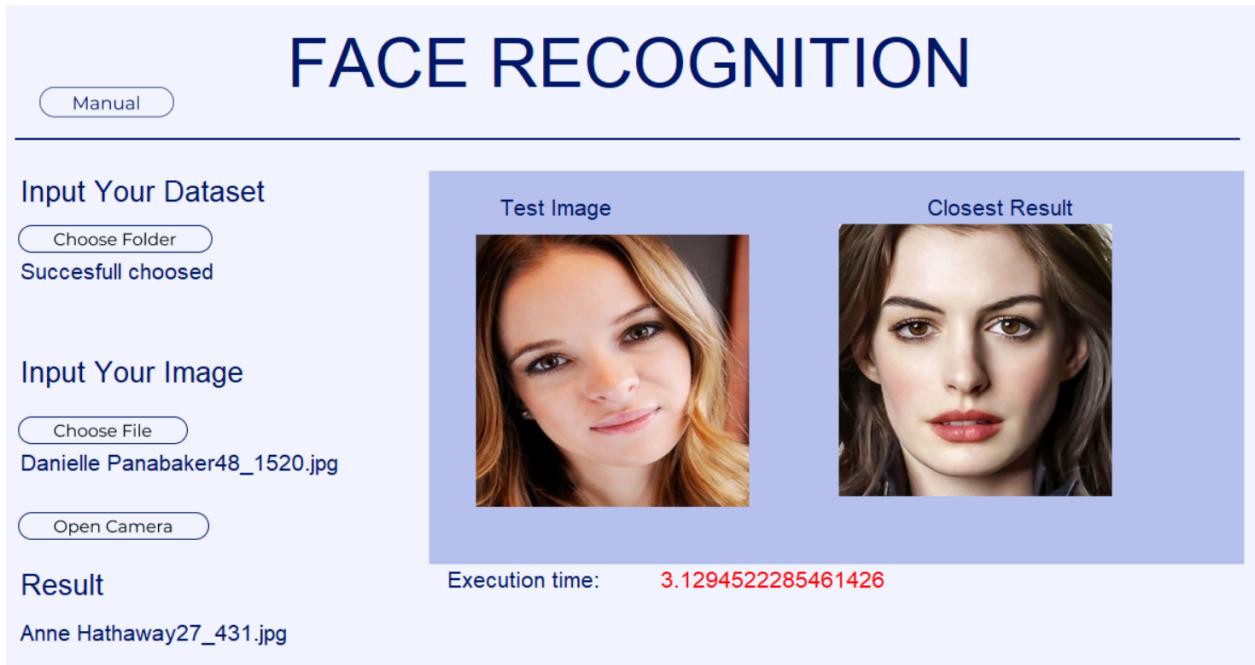
Open Camera

Result

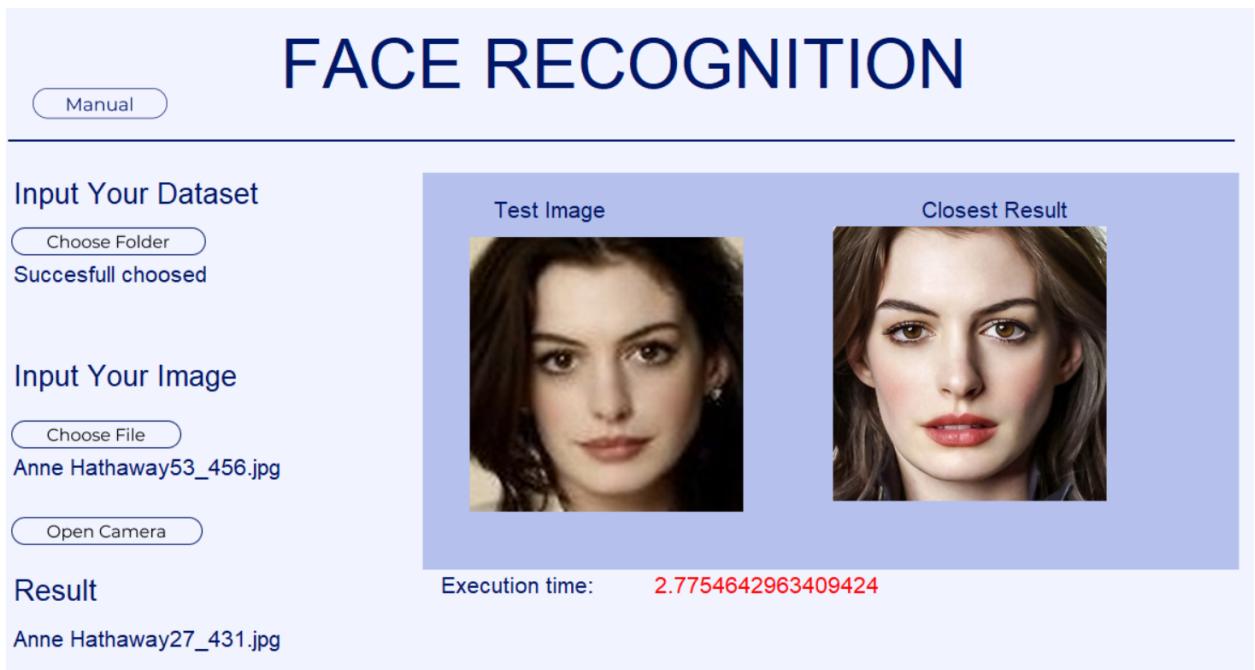
Anne Hathaway27_431.jpg



Gambar 4.1.4



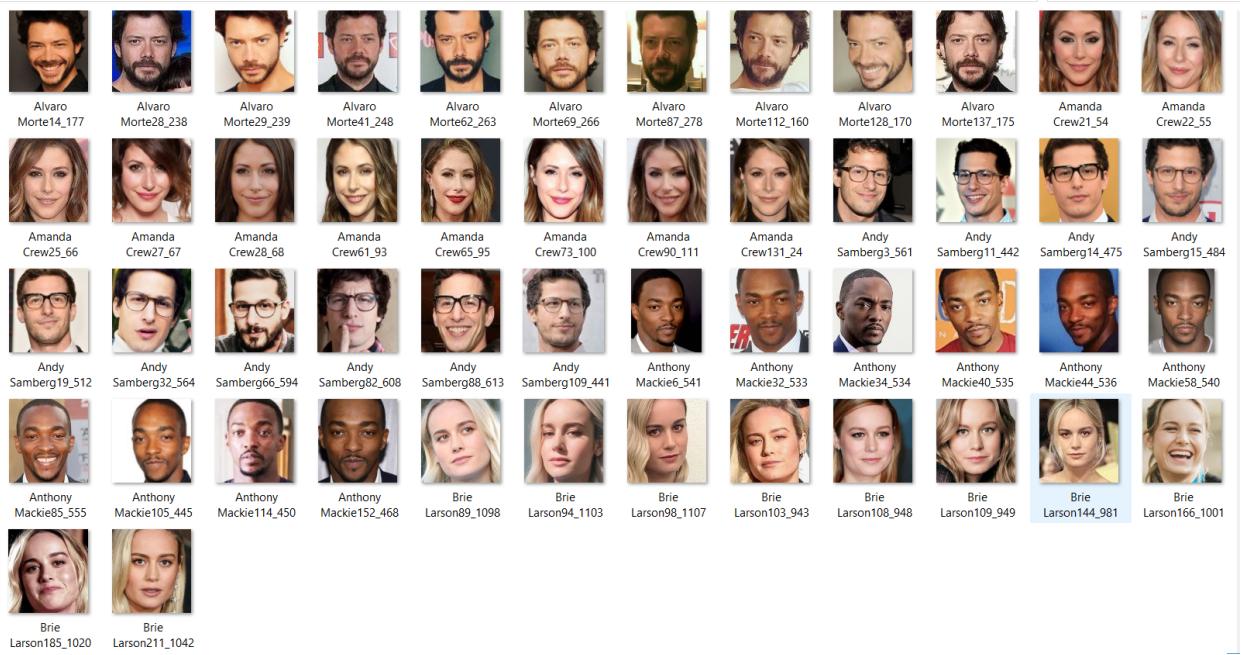
Gambar 4.1.5



Gamber 4.1.6

4.2 Eksperimen II

Pada eksperimen II, digunakan dataset 5 orang berbeda masing-masing 10 foto dengan total 50 foto.



Gambar 4.2.1 Dataset II

FACE RECOGNITION

Manual

Input Your Dataset

Choose Folder
Successfull choosed

Input Your Image

Choose File
Alvaro Morte52_255.jpg

Open Camera

Result

Alvaro Morte14_177.jpg

Test Image

Closest Result

Execution time: 8.130255460739136

Gambar 4.2.2



Gambar 4.2.3



Gambar 4.2.4

FACE RECOGNITION

Manual

Input Your Dataset

Choose Folder

Succesfull choosed

Input Your Image

Choose File

Andy Samberg26_557.jpg

Open Camera

Result

Alvaro Morte14_177.jpg



Gambar 4.2.5

FACE RECOGNITION

Manual

Input Your Dataset

Choose Folder

Succesfull choosed

Input Your Image

Choose File

Andy Samberg76_604.jpg

Open Camera

Result

Andy Samberg14_475.jpg



Gambar 4.2.6

4.3 Eksperimen III

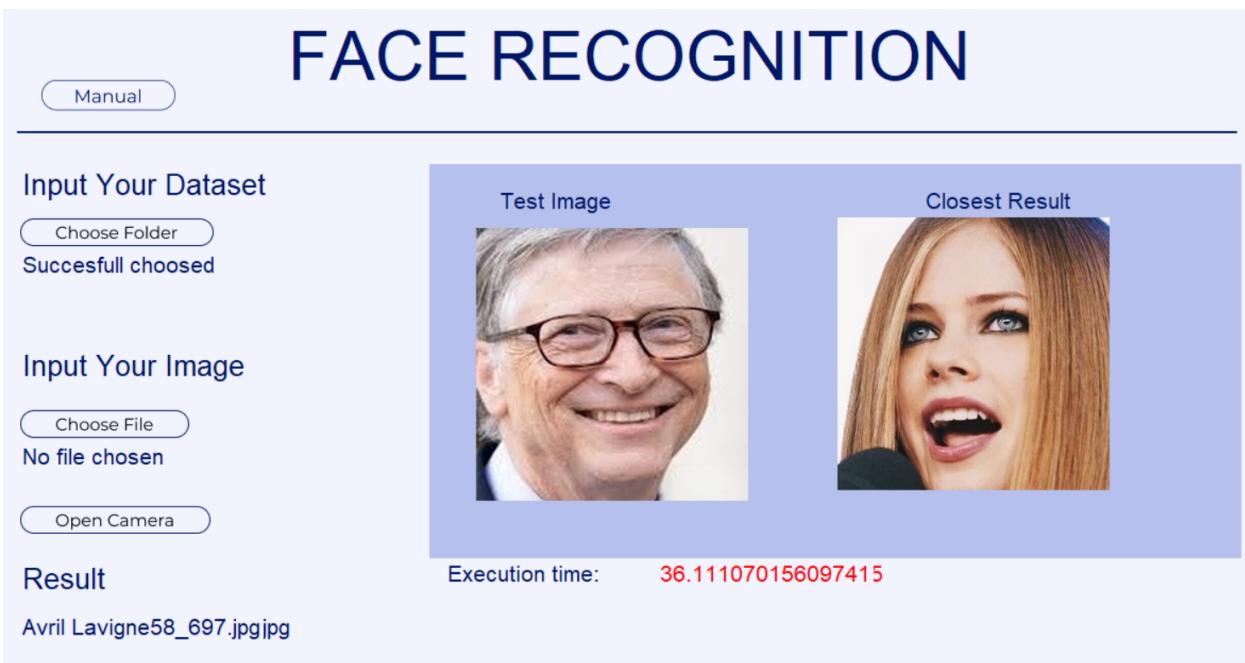
Pada eksperimen II, digunakan dataset 20 orang berbeda masing-masing 5 foto dengan total 100 foto.



Gambar 4.3.1 Dataset



Gambar 4.3.2



Gambar 4.3.3

FACE RECOGNITION

Manual

Input Your Dataset

[Choose Folder](#)
Successfull choosed

Input Your Image

[Choose File](#)
Amanda Crew25_66.jpg

[Open Camera](#)

Result

Elizabeth Lail0_1051.jpg pg

Test Image



Closest Result



Execution time: **36.088193655014045**

Gambar 4.3.4

FACE RECOGNITION

Manual

Input Your Dataset

[Choose Folder](#)
Successfull choosed

Input Your Image

[Choose File](#)
Anthony Mackie104_444.jpg

[Open Camera](#)

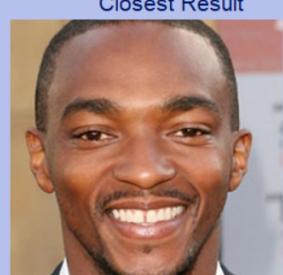
Result

Anthony Mackie92_559.jpg

Test Image



Closest Result



Execution time: **36.093075752258376**

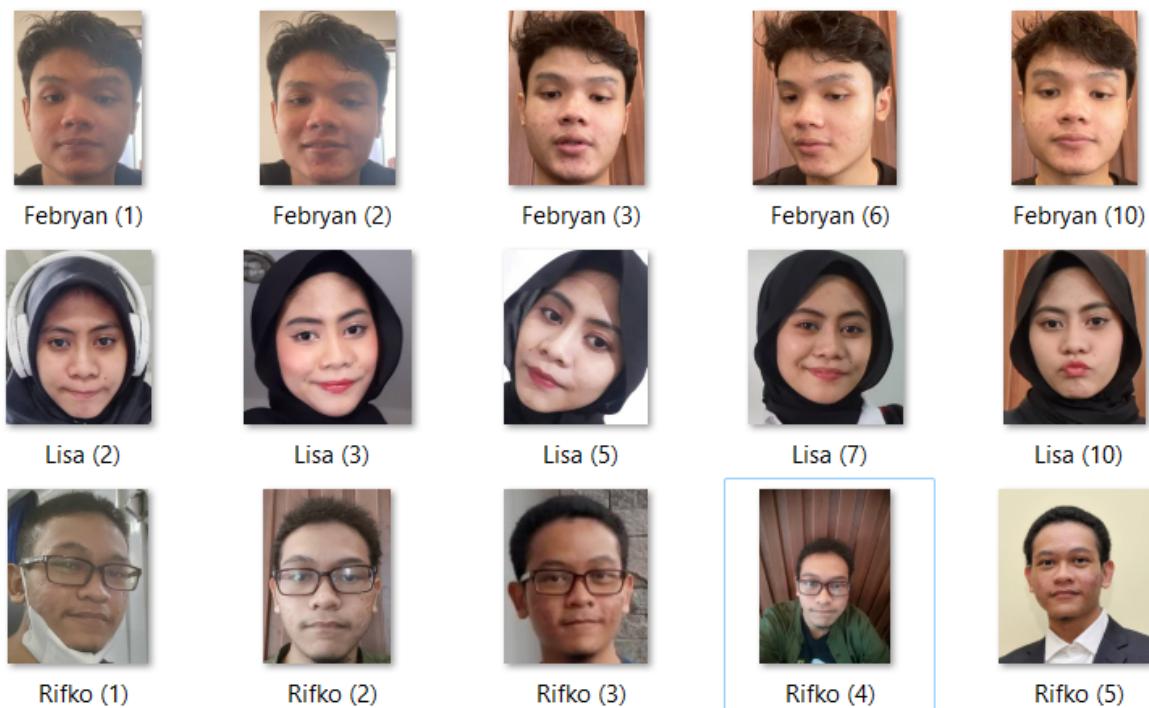
Gamber 4.3.5



Gambar 4.3.6

4.4 Eksperimen IV

Pada eksperimen IV, digunakan dataset dari anggota kelompok kami dengan masing-masing 5 foto dengan total 15 foto. Pada eksperimen ini pengenalan wajah dilakukan menggunakan kamera.



Gambar 4.4.1 Dataset IV



Gambar 4.4.2



Gambar 4.4.3

FACE RECOGNITION

Manual

Input Your Dataset

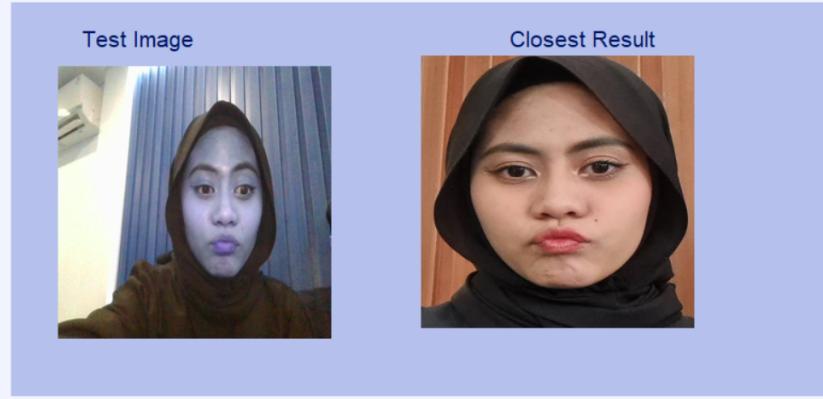
Succesfull choosed

Input Your Image

imagefromCam.png

Result

Lisa (10).jpg pg



Gambar 4.4.4

4.5 Keterangan

Eksperimen	Jumlah Data Set	Gambar	Waktu Eksekusi	Hasil
I	20	Gambar 4.1.2	2.8 detik	Sesuai
		Gambar 4.1.3	2.7 detik	Sesuai
		Gambar 4.1.4	2.8 detik	Tidak Sesuai
		Gambar 4.1.5	3.1 detik	Tidak Sesuai
		Gambar 4.1.6	2.7 detik	Sesuai
II	50	Gambar 4.2.2	8.1 detik	Sesuai
		Gambar 4.2.3	8.4 detik	Tidak Sesuai
		Gambar 4.2.4	8.1 detik	Sesuai
		Gambar 4.2.5	8.1 detik	Tidak Sesuai
		Gambar 4.2.6	8.6 detik	Sesuai
III	100	Gambar 4.3.2	35.9 detik	Sesuai
		Gambar 4.3.3	36.1 detik	Tidak Sesuai
		Gambar 4.3.4	36 detik	Tidak Sesuai
		Gambar 4.3.5	36 detik	Sesuai
		Gambar 4.3.6	36 detik	Tidak Sesuai
IV	15	Gambar 4.4.2	9.5 detik	Sesuai
		Gambar 4.4.3	7.15 detik	Sesuai
		Gambar 4.4.4	9.5 detik	Sesuai

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil eksperimen pada program, dapat disimpulkan bahwa program berjalan dengan cukup baik. Namun, masih banyak kesalahan pada pengenalan wajah, program seringkali memunculkan foto yang tidak sesuai. Ketidaksesuaian tersebut mungkin diakibatkan oleh beberapa faktor, salah satunya adalah kualitas gambar pada dataset (posisi wajah, *lighting*, dll). Faktor lainnya adalah ketidakpresision perhitungan dalam pengolahan data.

5.2 Saran

Hasil pengerjaan Tugas Besar 2 Aljabar Linier dan Geometri kami pastinya tidak luput dari hambatan dan kesalahan. Maka dari itu, kami telah mencatat beberapa saran atau poin penting bagi pihak pembaca yang berminat untuk membuat Face Recognition dengan menggunakan Eigenface:

- 1) Kurang akuratnya hasil ekstraksi foto dikarenakan hanya mengandalkan data foto tanpa mempertimbangkan jenis kelamin, usia, pencahayaan, kualitas foto, posisi wajah dan lain-lain.
- 2) Untuk pengenalan wajah, dapat dengan mencari dan menggunakan fungsi-fungsi library yang sudah ada agar perhitungan dapat menjadi lebih akurat dan hasil pengenalan wajah juga lebih akurat.
- 3)

5.3 Refleksi

Dengan menyelesaikan Tugas Besar 2 Aljabar Linier dan Geometri ini, penulis dapat memahami proses pengenalan wajah (*Face Recognition*) menggunakan metode Eigenface dengan memanfaatkan materi yang sudah dipelajari yaitu nilai eigen dan vektor eigen. Penulis menyadari bahwa di dalam pengerjaan ini masih terdapat banyak kekurangan dalam program yang penulis buat.

DAFTAR REFERENSI

- Ratz, Arthur V. 2021. *Can QR Decomposition Be Actually Faster? Schwarz-Rutishauser Algorithm.* <https://towardsdatascience.com/can-qr-decomposition-be-actually-faster-schwarz-rutishauser-algorithm-a32c0cde8b9b> diakses pada 16 November 2022.
- Anonymous. 2022. *Power iteration.* https://en.wikipedia.org/wiki/Power_iteration diakses 16 November 2022.
- Tam, Adrian. 2021. *Face Recognition using Principal Component Analysis.* <https://machinelearningmastery.com/face-recognition-using-principal-component-analysis/> diakses pada 13 November 2022.
- pawangfg. 2021. *ML | Face Recognition Using Eigenfaces (PCA Algorithm).* <https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/> diakses pada 13 November 2022.
- Munir, Rinaldi. 2020. *Nilai Eigen dan Vektor Eigen (Bagian 1).* <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf> diakses pada 3 November 2022.
- https://ergodic.ugr.es/cphys/LECCIONES/FORTRAN/power_method.pdf
<http://pi.math.cornell.edu/~web6140/TopTenAlgorithms/QRalgorithm.html>

LAMPIRAN

Link Github : <https://github.com/dhanikanovlisa/Algeo02-21075>

Link Video : <https://youtu.be/5uCy7zwLKHk>