

LAPORAN TUGAS BESAR IF2123

Aplikasi Nilai Eigen dan Eigenface Pada Pengenalan Wajah *(Face Recognition)*

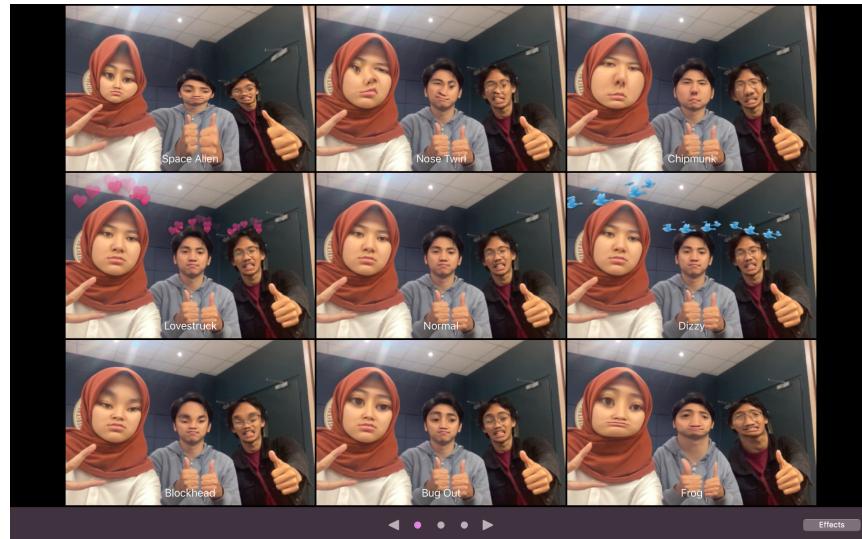
Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2123 Aljabar Linier dan Geometri pada Semester I Tahun Akademik 2022/2023

Disusun oleh:

Fajar Maulana Herawan (K1) 13521080

Aulia Mey Diva Annandya (K1) 13521103

M. Abdul Aziz Ghazali (K2) 13521128



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

2022

DAFTAR ISI

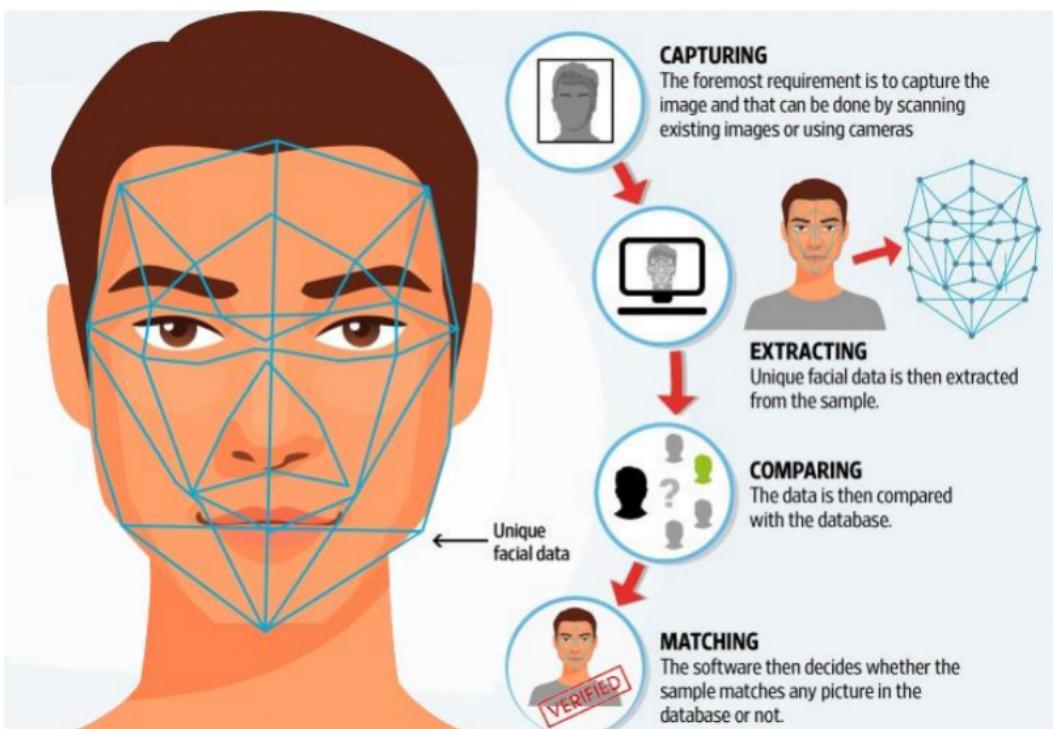
DAFTAR ISI	1
BAB I	3
BAB II	6
2.1 Perkalian Matriks	6
2.1.1 Tidak Komutatif	6
2.1.2 Distributif	7
2.1.3 Asosiatif	7
2.1.4 Perkalian dengan Skalar	7
2.1.5 Identitas	8
2.2 Nilai Eigen	8
2.3 Vektor Eigen	9
2.4 Eigenface	9
BAB III	11
3.1 Test Script	11
3.2 Implementasi Fungsi	13
3.2.1 getNorm	13
3.2.2 checkConverge	13
3.2.3 getEigenValueVector	14
3.2.4 gram_schmidt	14
3.2.5 convertImage	15
3.2.6 getMatrixCoef	15
3.2.7 getCoef	15
3.2.8 nearestDistance	15
3.2.9 cropimage	16
3.2.10 list_eigenface	16
3.2.11 cropAllImage	17
3.2.12 outputImage	17
3.2.13 trainingData	18
3.2.14 closestImage	18
3.2.15 displayEigenFace	19
3.2.16 convertFrame	19
3.2.17 cropframe	19
BAB IV	20

4.1 Berdasarkan Banyak Dataset	20
4.2 Berdasarkan Input Gambar	21
4.3 Berdasarkan Input Menggunakan Kamera.....	23
BAB V	25
5.1 Kesimpulan	25
5.2 Saran	25
5.3 Refleksi	25
REFERENSI	27
LAMPIRAN.....	28

BAB I

DESKRIPSI MASALAH

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 dibawah ini.



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.

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 tengah atau mean (Ψ) (2)
3. Langkah ketiga kemudian cari selisih (Φ) 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) (4)
5. Langkah kelima menghitung eigenvalue (λ) dan eigenvector (v) dari matriks kovarian (C)

$$C \mathbf{x} v_i = \lambda_i \mathbf{x} v_i \quad (5)$$

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

$$l = 1, \dots, M$$

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.

$$\varepsilon_k = \Omega - \Omega_k \quad (8)$$

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 II

TEORI SINGKAT

2.1 Perkalian Matriks

Perkalian matriks merupakan sebuah operasi dalam matriks dimana nilai pada matriks yang bisa dihasilkan dengan cara mengalikan tiap baris dengan setiap kolom yang memiliki jumlah baris yang sama. Setiap elemen matriks akan dikalikan dengan anggota elemen matriks lainnya. Syarat untuk melakukan perkalian matriks adalah matriks pertama harus memiliki jumlah kolom yang sama dengan jumlah baris pada matriks kedua.

$$A \times B$$
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} j & k \\ l & m \end{pmatrix}$$
$$A \times B = \begin{pmatrix} aj + bl & ak + bm \\ cj + dl & ck + dm \end{pmatrix}$$

Pada gambar diatas dapat terlihat bahwa matriks A memiliki dua baris dan dua kolom, begitu juga dengan matriks B memiliki dua baris dan dua kolom. Kedua matriks tersebut memenuhi syarat perkalian matriks. Perkalian matriks juga memiliki sifat-sifat yang bisa diterapkan, antara lain:

2.1.1 Tidak Komutatif

$$AB \neq BA$$

Dengan A dan B adalah matriks. Jika A berukuran $m \times n$ dan B berukuran $p \times q$, maka AB terdefinisi ketika $n = p$, dan BA terdefinisi ketika $m = q$. Secara umum, sifat komutatif ini tidak berlaku pada perkalian matriks. Hanya pada kasus $m = q = n = p$,

atau ketika A dan B merupakan matriks persegi dengan ukuran yang sama, maka kedua perkalian matriks akan terdefinisi.

2.1.2 Distributif

$$A(B+C) = AB + AC$$

$$(B+C)D = DB + CD$$

Dengan A, B, dan C merupakan matriks yang masing-masing berukuran $m \times p$, $p \times n$, dan $p \times n$. Pada rumus pertama di atas, sifat distributif mengartikan matriks memiliki sifat distributif kiri, sedangkan pada rumus pertama di atas, sifat distributif mengartikan matriks memiliki sifat distributif kanan.

2.1.3 Asosiatif

$$(AB)C = A(BC)$$

Dengan A, B, dan C merupakan matriks. Hasil perkalian di atas akan terdefinisi jika banyak kolom di A sama dengan banyak baris di B dan banyak kolom di B sama dengan banyak baris di C.

2.1.4 Perkalian dengan Skalar

$$c(AB) = (cA)B \text{ dan } (AB)c = A(Bc)$$

Dengan A dan B adalah matriks dan c adalah sebuah skalar. Dalam kasus c merupakan scalar bersifat komutatif dan perkalian AB terdefinisi, dalam artian banyak kolom di A sama dengan banyak baris di B, maka perkalian matriks dengan skalar di atas terdefinisi.

2.1.5 Identitas

$$AI = IA = A$$

Dengan A adalah matriks dan I adalah matriks identitas.

2.2 Nilai Eigen

Nilai Eigen adalah nilai karakteristik dari suatu matriks berukuran nxn. Misal sebuah matriks A berukuran nxn dan x adalah vektor tidak-nol. Jika Ax sama dengan perkalian suatu skalar dengan x, maka merupakan nilai eigen dari A.

$$Ax = \lambda x$$

$$IAx = \lambda Ix$$

$$Ax = \lambda Ix$$

$$(\lambda I - A)x = 0$$

Agar memiliki solusi tidak-nol, maka nilai eigen dicari melalui persamaan $\det(\lambda I - A) = 0$. Persamaan determinan tersebut adalah persamaan karakteristik dari matriks A dan akar akar persamaannya, atau yang kita sebut , disebut akar-akar karakteristik atau nilai-nilai eigen. Contoh soal mencari nilai eigen:

Nilai Eigen (λ)

$$(\lambda I - A)v = 0 \quad \leftarrow \text{Persamaan Karakteristik}$$

$$\left(\lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & -2 \\ 1 & 4 \end{bmatrix} \right) v = 0 \quad \leftarrow \text{Matriks } I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\left(\begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 1 & -2 \\ 1 & 4 \end{bmatrix} \right) v = 0$$

$$\left(\begin{bmatrix} \lambda - 1 & 2 \\ -1 & \lambda - 4 \end{bmatrix} \right) v = 0$$

Untuk menentukan nilai λ yang skalar, berlaku:

$$\det(\lambda I - A) = 0$$

$$\det \begin{bmatrix} \lambda - 1 & 2 \\ -1 & \lambda - 4 \end{bmatrix} = 0 \quad \leftarrow \text{Determinan Matriks ordo } 2 \times 2 = (ad - bc)$$

$$((\lambda - 1)(\lambda - 4) - (2)(-1)) = 0$$

$$(\lambda^2 - 5\lambda + 4 + 2) = 0$$

$$\lambda^2 - 5\lambda + 6 = 0$$

$$(\lambda - 2)(\lambda - 3) = 0$$

$$\lambda = 2 \text{ atau } \lambda = 3 \quad \leftarrow \text{Nilai Eigen}$$

2.3 Vektor Eigen

Vektor Eigen adalah vektor kolom bukan nol yang bila dikalikan dengan sebuah matriks berukuran $n \times n$ akan menghasilkan vektor lain yang memiliki nilai kelipatan dari vektor Eigen itu sendiri. Seperti rumus dari nilai eigen yaitu, x merepresentasikan vektor eigen yang berkorespondensi dengan λ . Cara menghitung vektor eigen adalah dengan menemukan nilai eigen dari matriks A . Kemudian, untuk setiap nilai eigen yang dimiliki, temukan x dengan eliminasi Gauss untuk membentuk matriks eselon. Vektor eigen yang terkait dengan masing-masing nilai eigen sudah didapatkan. Contoh soal mencari vektor eigen:

Vektor Eigen
 $(\lambda I - A)v = 0$
 $([\begin{array}{cc} \lambda - 1 & 2 \\ -1 & \lambda - 4 \end{array}] v) = 0$
 $[\begin{array}{cc} \lambda - 1 & 2 \\ -1 & \lambda - 4 \end{array}] [\begin{array}{c} v_1 \\ v_2 \end{array}] = 0$

Untuk nilai $\lambda = 2$, maka:
 $[\begin{array}{cc} 1 & 2 \\ -1 & -2 \end{array}] [\begin{array}{c} v_1 \\ v_2 \end{array}] = [\begin{array}{c} 0 \\ 0 \end{array}]$

Terbentuk Sistem Persamaan Linear
 $v_1 + 2v_2 = 0$
 $-v_1 - 2v_2 = 0$

Diperoleh:
 $v_1 = -2$ dan $v_2 = 1$

Pembuktian:
 $Av = \lambda v$
 $[\begin{array}{cc} 1 & -2 \\ 1 & 4 \end{array}] [\begin{array}{c} -2 \\ 1 \end{array}] = 2 [\begin{array}{c} -2 \\ 1 \end{array}]$
 $[\begin{array}{c} -4 \\ 2 \end{array}] = 2 [\begin{array}{c} -2 \\ 1 \end{array}]$ terbukti

Untuk nilai $\lambda = 3$, maka:
 $[\begin{array}{cc} 2 & 2 \\ -1 & -1 \end{array}] [\begin{array}{c} v_1 \\ v_2 \end{array}] = [\begin{array}{c} 0 \\ 0 \end{array}]$

Terbentuk Sistem Persamaan Linear
 $2v_1 + 2v_2 = 0$
 $-v_1 - v_2 = 0$

Diperoleh:
 $v_1 = 1$ dan $v_2 = -1$

Pembuktian:
 $Av = \lambda v$
 $[\begin{array}{cc} 1 & -2 \\ 1 & 4 \end{array}] [\begin{array}{c} 1 \\ -1 \end{array}] = 3 [\begin{array}{c} 1 \\ -1 \end{array}]$
 $[\begin{array}{c} 3 \\ -3 \end{array}] = 3 [\begin{array}{c} 1 \\ -1 \end{array}]$ terbukti

2.4 Eigenface

Eigenface adalah salah satu algoritma yang cukup popular dalam menyelesaikan masalah pengenalan wajah. Eigenface merupakan sekumpulan eigen vektor yang berasal dari kovarian matriks dengan distribusi acak pada citra wajah dimensi tinggi. Eigenface digunakan untuk mereduksi dimensi vector wajah menjadi vektor yang lebih sederhana (eigen vektor). Langkah-langkah pada algoritma eigenface:

1. Mengubah gambar wajah menjadi matriks.
2. Mencari *mean* dari keseluruhan matriks.

3. Mencari normalized dengan menghitung selisih nilai matriks rata-rata dengan keseluruhan gambar.
4. Setelahnya, menghitung matriks kovarian.
5. Menghitung eigen value dan eigen vektor dari matriks kovarian dengan rumus $C \mathbf{x} \nu_i = \lambda_i \mathbf{x} \nu_i$.
6. Menghitung eigenface.

BAB III

IMPLEMENTASI PUSTAKA

3.1 Test Script

No.	Nama Fungsi	Deskripsi dan tujuan fungsi	Parameter yang dibutuhkan	Hasil dari Fungsi
1.	getNorm	Mendapatkan norm dari matriks	- m (m adalah sebuah matriks)	\mengembalikan sebuah value berupa norm dari matriks
2.	checkConverge	Mengecek kekonvergenan elemen matriks	- arr - arr_prev	Mengembalikan true jika matriks konvergen terhadap suatu nilai dan mengembalikan false jika matriks tidak konvergen terhadap suatu nilai
3.	getEigenValueVector	QR dekomposisi matriks untuk mencari eigenvalue dan eigenvector	-A (sebuah matriks yang akan konvergen menuju matriks)	Mengembalikan copyMatrix matriks yang diagonalnya eigenvalue dan QQ adalah eigenvektor
4.	gram_schmidt	Algoritma QR menggunakan metode gram schmidt	-A(sebuah matriks yang akan didekomposisi)	Mengembalikan hasil dekomposisi matriks A
5.	convertImage	Mengubah image menjadi matriks 256*256x1	-imagename(file image yang akan di convert)	Mengembalikan sebuah matriks hasil convert image
6.	getMatrixCoef	Menghasilkan list kombinasi linier dari normalized matriks yang ada pada dataset	-bestEigenVector(beberapa eigenvector terbaik) -normalizedDataSet(Matriks normalized dataset)	Mengembalikan list kombinasi linier yang berbentuk (<kumpulan eigenface>, <kumpulan gambar>)

7.	getCoef	Menghasilkan kombinasi linier dari normalized matriks	-list_bestEigenface (List berisi eigenvector ter) -vectorImageInput	Mengembalikan kombinasi linier dari normalized matriks, kombinasi linier berbentuk (<kumpulan eigenface>,1)
8.	nearestDistance	Mengeluarkan X minimum antara gambar input dengan gambar di dataset	-InputCoef(Matriks kombinasi linier dari gambar input) -MatrixCoef (Matriks kombinasi linier dari dataset)	Mengembalikan hasil minimum selisih gambar input dan dataset
9.	cropimage	Memotong gambar agar berbentuk persegi	-image(file image yang akan di crop)	Mengembalikan hasil image yang sudah ter-crop
10.	list_eigenface	Menghasilkan Matriks EigenFace	-normalizedMatrix (Matriks normalized) -all_image(matriks dari semua image)	Mengembalikan Matriks EigenFace dengan ukuran (256*256, <kumpulan eigenface terbaik>)
11.	cropAllImage	Memotong semua gambar	-path (path folder yang berisi semua gambar yang akan di-crop)	Folder yang berisi semua gambar telah terpotong menjadi persegi
12.	outputImage	Mengeluarkan gambar dataset yang paling mirip dengan gambar input	-dirPath() -InputCoef(Matriks kombinasi linier dari gambar input) -MatrixCoef (Matriks kombinasi linier dari dataset)	Mengeluarkan gambar di dataset yang paling dengan gambar input
13.	trainingData	Menghasilkan matriks eigenface dan matriks koefisien	-path(path folder)	Matriks eigenface dan matriks koefisien terbentuk
14.	closestImage	Mengeluarkan gambar yang paling mirip	-path(path folder dataset) -InputCoef(Matriks kombinasi linier	Mengeluarkan gambar yang paling mirip jika tidak ada keluarkan "Gambar tidak ditemukan"

			dari gambar input) -MatrixCoef (Matriks kombinasi linier dari dataset)	
15.	displayEigen Face	Menampilkan eigenface	eigennface(kumpulan hasil eigenface)	Menampilkan gambar-gambar dari eigenface
16.	convertFrame	Mengubah frame image menjadi matriks 256*256,1	frame(frame image yang akan convert)	Mengembalikan matriks frame yang telah terubah
17.	cropframe	Crop frame image di wajah user	-frame(frame image yang akan convert) -path(path folder tempat hasil crop akan disimpan)	Menghasilkan frame image yang sudah tercrop

3.2 Implementasi Fungsi

3.2.1 getNorm

```
def getNorm(m):
    '''Mendapatkan norm dari matriks'''
    matrix = np.square(m)
    matrix = np.sum(matrix)
    matrix = np.sqrt(matrix)

    return matrix
```

3.2.2 checkConverge

```
def checkConverge(arr,arr_prev):
    ''' Mengecek nilai elemen matriks apakah konvergen menuju suatu nilai '''
    for i in range(len(arr)):
        for j in range(len(arr)):
            if (getNorm(arr_prev-arr)> 10000): # Parameter dapat diubah sesuai tingkat akurasi
                return False
    return True
```

3.2.3 getEigenValueVector

```
def getEigenValueVector(A):
    ''' Algoritma QR untuk mencari eigenvalue dan eigenvector , dimana matriks Ak akan konvergen menuju matriks schur '''
    copy (variable) copyMatrix: Any
    n = copyMatrix.shape[0]
    QQ = np.eye(n)
    i = 0
    while(True):
        copyMatrix_copy = np.copy(copyMatrix)
        s = copyMatrix.item(n-1, n-1)
        smult = s * np.eye(n)
        # Dengan metode gram schmidt dilakukan dekomposisi QR
        Q, R = gram_schmidt((copyMatrix - smult))
        # smult dikembalikan nilainya
        copyMatrix = np.add(R @ Q, smult)
        QQ = QQ @ Q
        i += 1
        if(checkConverge(copyMatrix,copyMatrix_copy)):
            # Cek parameter konvergen
            break
    return copyMatrix,QQ # QQ adalah eigenvector , copyMatrix adalah matrix yang diagonalnya eigenvalue
```

3.2.4 gram_schmidt

```
def gram_schmidt(A):
    '''Algoritma QR menggunakan metode gram schmidt, mengeluarkan hasil dekomposisi matriks A'''
    if len(A) != 0 :
        Q = np.empty([len(A), len(A)])
        counter = 0

        # Mencari Matrix Orthogonal
        for a in A.T:

            u = np.copy(a)
            for i in range(0, counter):
                proj = np.dot(np.dot(Q[:, i].T , a) , Q[:, i])
                u -= proj

            e = u / getNorm(u)
            Q[:, counter] = e

            counter += 1

        # menghitung matriks segitiga atas
        R = np.dot(Q.T, A)

        return Q, R
    else:
        print("Image tidak kedetect")
```

3.2.5 convertImage

```
def convertImage(imagename):
    '''Mengubah image menjadi matriks n*n x 1'''
    image = cv2.imread(imagename)
    image = cv2.resize(image, (256, 256), interpolation = cv2.INTER_AREA)
    greyscaleimg = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    converted = greyscaleimg.flatten()
    return converted
```

3.2.6 getMatrixCoef

```
def getMatrixCoef(bestEigenVector, normalizedDataSet) :
    '''Menghasilkan List kombinasi linier dari normalized matriks yang ada pada dataset
       ,List ini berbentuk (<banyak eigenface>,<banyak gambar>)'''
    CoefOfLinComMatrix = np.empty((len(bestEigenVector[0]),0), float)
    for i in range(len(normalizedDataSet[0])) :
        LinComOfNormalized = getCoef(bestEigenVector, np.transpose([normalizedDataSet[:,i]]))
        CoefOfLinComMatrix = np.column_stack((CoefOfLinComMatrix, LinComOfNormalized))
    return CoefOfLinComMatrix
```

3.2.7 getCoef

```
def getCoef(list_bestEigenface, vectorImageInput) :
    '''Menghasilkan kombinasi linier dari normalized matriks, (<banyak eigenface>,1)'''
    matrixCoef = np.transpose([np.linalg.lstsq(list_bestEigenface, np.transpose(vectorImageInput)[0], rcond=None)[0]])
    return matrixCoef
```

3.2.8 nearestDistance

```
def nearestDistance(InputCoef, MatrixCoef) :
    '''Mengeluarkan nilai X minimum antara gambar input dengan gambar di data set'''
    '''X = |I-M|, I = Matriks kombinasi linear dari input dan M = List Matriks kombinasi linear dari gambar di dataset'''
    minimum = getNorm(np.subtract(InputCoef, np.transpose([MatrixCoef[:, 0]])))
    for i in range(len(MatrixCoef[0])) :
        e = getNorm(np.subtract(InputCoef, np.transpose([MatrixCoef[:, i]])))
        if (e < minimum) :
            minimum = e
    return minimum
```

3.2.9 cropimage

```
def cropimage(image):
    '''Mengcrop gambar di wajah pengguna'''
    frame = cv2.imread(image)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier('src/src_feature/face.xml')
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    if len(faces) > 0:
        for (x, y, w, h) in faces:
            crop_img = frame[y:y+h+10, x:x+w+10]
            cv2.imwrite(image, crop_img)
            return image
    else:
        print("Tidak ada")
        os.remove(image)
        return image
```

3.2.10 list_eigenface

```
def list_eigenface(normalizedMatrix,all_image):
    '''Menghasilkan Matriks EigenFace dengan ukuran (256*256,<banyak eigenvector terbaik>)'''
    print("Mencari eigenface")
    Covariance = np.matmul(np.transpose(normalizedMatrix),normalizedMatrix) #A'A = (<banyak gambar>, <banyak gambar>
    eigenvalue, eigenvector = getEigenValueVector(Covariance)
    bestEigenVector = np.empty((len(eigenvector[0]),0), float)
    counter = 0
    for i in range(len(eigenvector[0])) :
        if (eigenvalue[i][i] > 1) : # Tuning , Memilih eigenvector yang eigenvaluenya yang tidak koma
            bestEigenVector = np.column_stack((bestEigenVector, np.transpose([eigenvector[:, i]])))
            # break
            counter += 1
    # bestEigenVector = eigenvector[:, :counter]
    list_bestEigenface = np.empty((256*256, 0), float)
    for i in range(len(bestEigenVector[0])) :
        eigenface = np.matmul(all_image, np.transpose([bestEigenVector[:, i]]))
        list_bestEigenface = np.column_stack((list_bestEigenface, eigenface))

    return list_bestEigenface
#ini disave
```

3.2.11 cropAllImage

```
def cropAllImage(path):
    # Mengcrop semua image
    print("Mengcrop semua image")
    folderpath = path
    print(folderpath)
    for (dirPath, dirNames, file) in os.walk(folderpath):
        for fileNames in file :
            tempPath = os.path.join(dirPath, fileNames)
            cropimage(tempPath)
    print("Selesai crop semua image")
```

3.2.12 outputImage

```
def outputImage (dirPath, MatrixCoef, InputCoef) :
    '''Mengeluarkan gambar yang paling mirip dengan gambar input di dataset'''
    minimum = getNorm(np.subtract(InputCoef, np.transpose([MatrixCoef[:, 0]])))
    index = 0
    imageOrder = 1
    for i in range(len(MatrixCoef[0])) :
        index += 1
        distance = getNorm(np.subtract(InputCoef, np.transpose([MatrixCoef[:, i]])))
        if (distance < minimum) :
            minimum = distance
            imageOrder = index
    count = 0
    for (dirPath, dirNames, file) in os.walk(dirPath):
        for fileNames in file :
            count += 1
            if count == imageOrder :
                return os.path.join(dirPath, fileNames)
```

3.2.13 trainingData

```
def trainingData(path):
    '''Menghasilkan matriks eigenface dan matriks koefisien'''
    folderpath = path
    A = np.empty((256*256,0), float) #Matriks Gambar (256*256,<Banyak Gambar>)
    for (dirPath, dirNames, file) in os.walk(folderpath):
        for fileNames in file :
            tempPath = os.path.join(dirPath, fileNames)
            convertedImage = convertImage(tempPath)
            A= np.column_stack((A, convertedImage.reshape(256*256, 1)))

    normalizedMatrix = A - A.mean(axis=1, keepdims=True)
    eigeface = list_eigenface(normalizedMatrix,A)
    MatrixCoef = getMatrixCoef(eigeface, normalizedMatrix)
    np.savetxt(f"src/data/matriksCoef.txt", MatrixCoef, delimiter=";")
    np.savetxt(f"src/data/eigenface.txt", eigeface, delimiter=";")
    print("Training Selesai")
    return MatrixCoef, eigeface
```

3.2.14 closestImage

```
def closestImage(path, InputCoef, MatrixCoef):
    '''Mengeluarkan Gambar yang paling mirip jika tidak ada keluarkan "Gambar tidak ditemukan" '''
    print("Mencari gambar terdekat")
    folderpath = path
    minimum = nearestDistance(InputCoef, MatrixCoef)
    print(minimum, "min")
    if minimum > 1.5 :    # Tuning minimum
        print("Gambar terdekat")
        nearestImage =  outputImage(folderpath, MatrixCoef, InputCoef)
        print(nearestImage)
        return nearestImage
    else:
        nearestImage =  outputImage(folderpath, MatrixCoef, InputCoef)
        print(nearestImage)
        print("Gambar tidak ditemukan")
        return None
```

3.2.15 displayEigenFace

```
def displayEigenFace(eigenFace) :  
    '''Menampilkan Gambar EigenFace'''  
    print("Menampilkan eigenface")  
    fig = plt.figure()  
    for i in range(len(eigenFace[0])) :  
        fig.add_subplot(10, 16, i+1)  
        plt.imshow(eigenFace[:, i].reshape(256, 256), cmap='gray')  
    plt.show()
```

3.2.16 convertFrame

```
def convertFrame(frame):  
    '''Mengubah Frame image menjadi matriks n*n x 1'''  
    frame = cv2.resize(frame, (256, 256), interpolation = cv2.INTER_AREA)  
    greyscaleimg = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  
    converted = greyscaleimg.flatten()  
    return converted
```

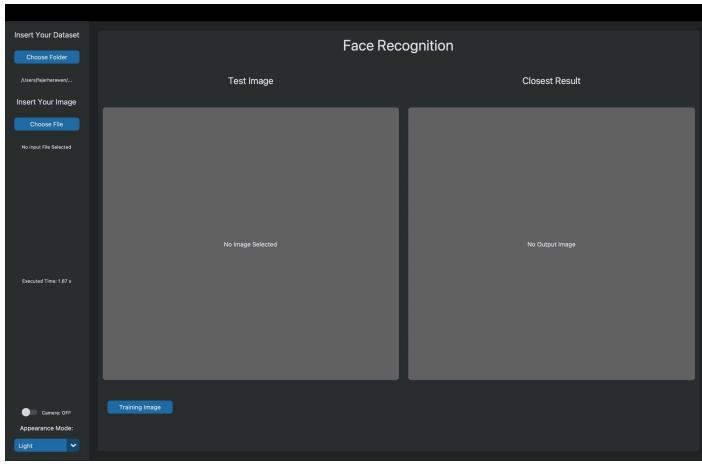
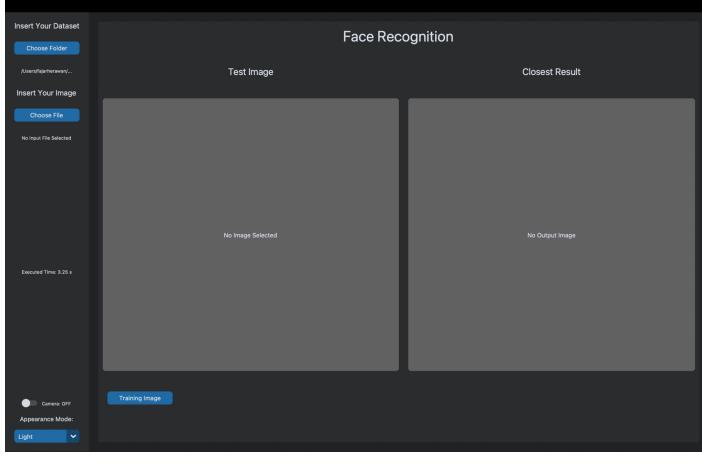
3.2.17 cropframe

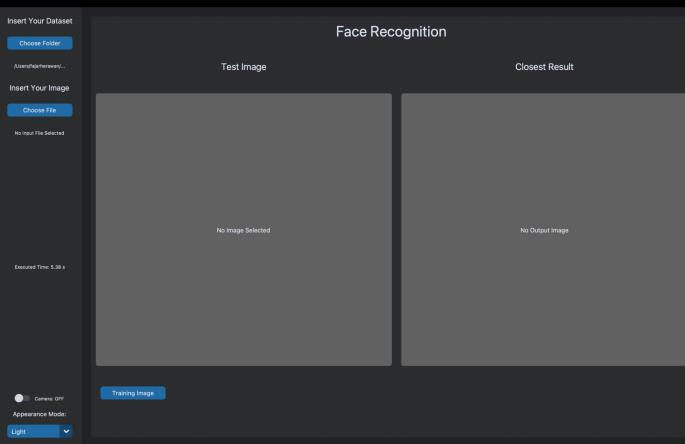
```
def cropframe(frame,path):  
    '''Mengcrop gambar di wajah pengguna'''  
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  
    face_cascade = cv2.CascadeClassifier('src\src_feature\face.xml')  
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)  
    if len(faces) > 0:  
        for (x, y, w, h) in faces:  
            crop_img = frame[y:y+h+10, x:x+w+10]  
            cv2.imwrite(path, crop_img)
```

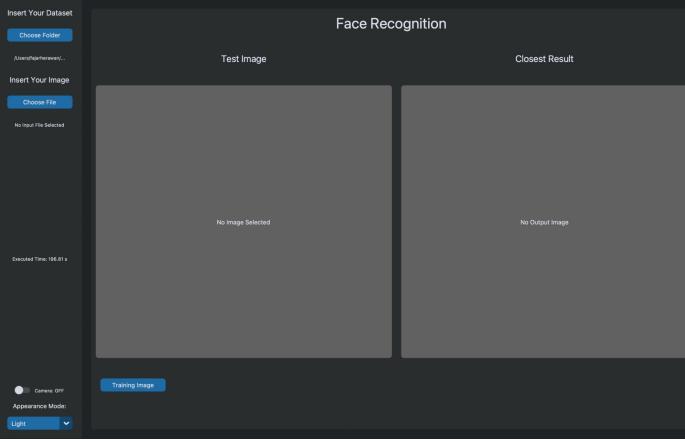
BAB IV

EKSPERIMEN

4.1 Berdasarkan Banyak Dataset

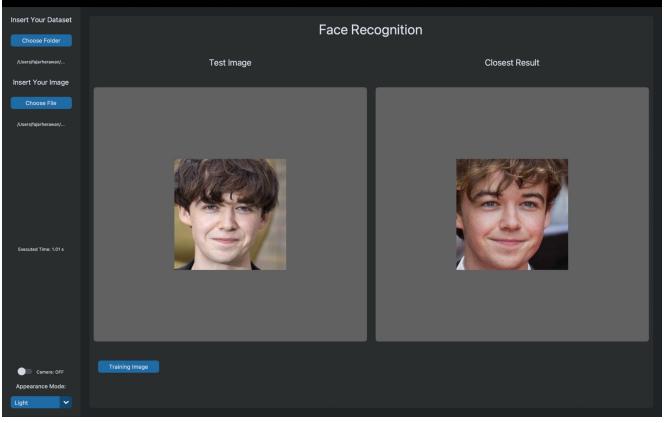
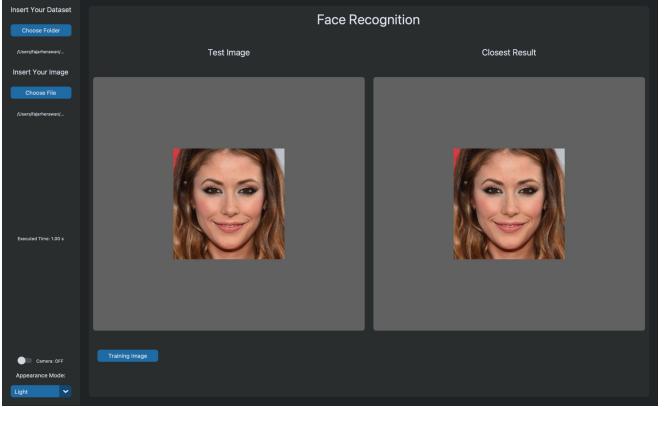
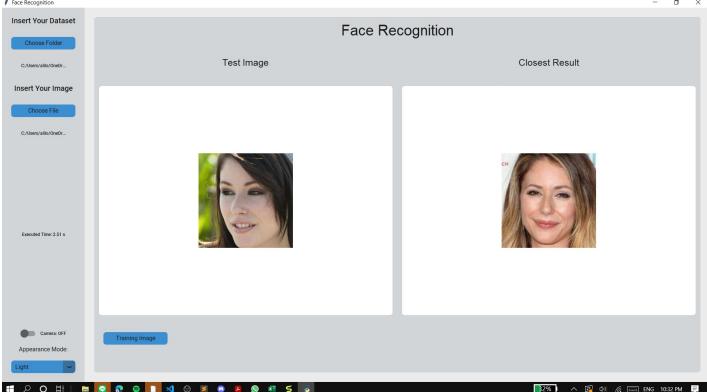
Tampilan	Deskripsi
	Jumlah Dataset : 24 Executed Time : 1.67s Deskripsi : Waktu yang dibutuhkan untuk training 24 data adalah 1.67s
	Jumlah Dataset : 48 Executed Time : 3.25s Deskripsi : Waktu yang dibutuhkan untuk training 48 data adalah 3.25s

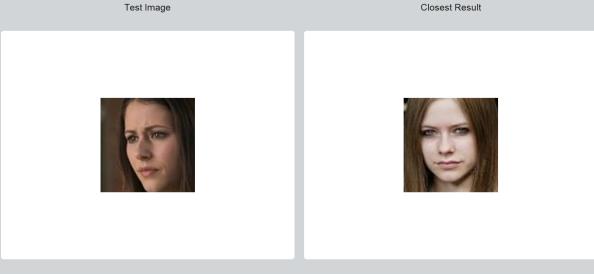
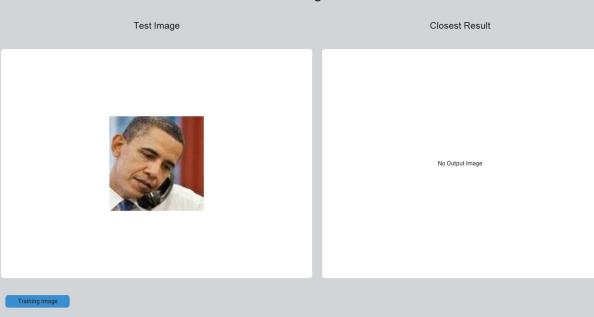
	<p>Jumlah Dataset : 96 Executed Time : 5.38s Deskripsi : Waktu yang dibutuhkan untuk training 96 data adalah 5.38s</p>
---	--

	<p>Jumlah Dataset : 200 Executed Time : 196.81s Deskripsi : Waktu yang dibutuhkan untuk training 200 data adalah 196.81s</p>
--	--

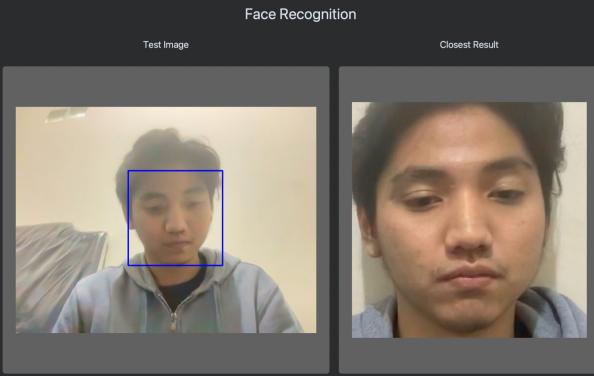
4.2 Berdasarkan Input Gambar

Tampilan	Deskripsi
----------	-----------

	<p>Executed Time : 1.01s</p> <p>Deskripsi : Saat gambar yang diinput tidak ada dalam dataset, output yang dikeluarkan adalah gambar yang paling mirip (orang yang sama).</p>
	<p>Executed Time : 1.00s</p> <p>Deskripsi : Saat gambar yang diinput ada dalam dataset, output yang dikeluarkan adalah gambar yang sama persis.</p>
	<p>Executed Time : 2.51s</p> <p>Deskripsi : Output yang dikeluarkan berbeda karena arah wajah input gambar dan dataset yang berbeda disebabkan orientasi wajah yang input dan dataset tidak sama.</p>

	<p>Execution time : 2.58s</p> <p>Deskripsi : Dengan input gambar wajah yang berbeda orientasi (agak miring) maka dapat pula mendapat hasil yang tidak akurat. Ini disebabkan karena eigenface didesain untuk menyimpan feature wajah yang orientasi lurus ke depan, sehingga kurang bisa merepresentasikan wajah yang berbeda arah.</p>
	<p>Execution time : 0s</p> <p>Deskripsi : Result akan menampilkan “No Output Image” jika input gambar wajah yang dimasukkan, wajah yang sama tidak terdapat dalam dataset yang digunakan.</p>

4.3 Berdasarkan Input Menggunakan Kamera

Tampilan	Deskripsi
	<p>Execution time : 0.73 s</p> <p>Deskripsi : Kamera mendeteksi wajah serta mengambil gambar dan dicari kemiripan wajah dengan gambar yang berada di dataset</p>

The screenshot shows a Face Recognition application window. On the left, there's a sidebar with buttons for 'Insert Your Dataset' (Choose Folder, Insert Your Image, Choose File), 'Execution Time: 0.78 s', and camera controls (Camera ON, Appearance Mode: Light). The main area has two panels: 'Test Image' showing a person with glasses and a red shirt, and 'Closest Result' showing a close-up of the same person's face. At the bottom, there are buttons for 'Training Image' and 'Light'.

Execution time : 0.78 s

Deskripsi : Kamera mendeteksi wajah serta mengambil gambar dan dicari kemiripan wajah dengan gambar yang berada di dataset

BAB V

KESIMPULAN

5.1 Kesimpulan

Pengenalan wajah manusia adalah sebuah teknologi untuk mengidentifikasi dan mengkonfirmasi identitas seseorang dengan wajahnya. Pengenalan wajah atau yang biasa disebut *face recognition* ini dapat dibuat dengan menggunakan algoritma *eigenface*. *Eigenface* dibentuk dengan mencari *eigenvalue* dan *eigenvector* nya. Di tugas besar Algeo 2 ini, kami membuat program face recognition yang dapat mendeteksi kemiripan input gambar dengan gambar yang berada di dataset. Kami menggunakan python untuk membuat program ini dengan menggunakan algoritma eigenface. *Eigenface* dibentuk dengan mencari *eigenvalue* dan *eigenvector*. Untuk mencari *eigenvalue* dan *eigenvector*, kami menggunakan dekomposisi QR dengan metode gram schmidt. Program *face recognition* ini ditampilkan dalam bentuk aplikasi dengan Tkinter. Aplikasi kami juga menyediakan fitur kamera yang dapat mendeteksi kemiripan wajah dengan dataset secara *real-time*. Dengan begitu, dapat disimpulkan bahwa *face recognition* dapat dibuat dengan *eigenface*.

5.2 Saran

Setelah mengerjakan tugas besar kedua ini, kami memiliki beberapa saran pada program *face recognition* kami karena belum tercapai pengenalan wajah dalam kondisi yang minim cahaya dan arah wajah yang berbeda. Hal tersebut disebabkan karena nilai eigenface dari gambar berpengaruh dengan intensitas cahaya. Selain itu, belum tercapai modularisasi yang sempurna. Oleh karena itu, jika diberikan kerangka program yang lebih terstruktur bisa membuat waktu pengerjaan lebih singkat dan waktu untuk optimalisasi program lebih banyak.

5.3 Refleksi

Tugas besar kedua ini , kami merasa *time management* kami masih kurang disebabkan beberapa tugas besar juga diberikan dalam kurun waktu yang singkat. Dalam tugas besar kedua

ini, kami belajar bukan hanya dalam pemrograman ,melainkan juga mengkoordinasi waktu dan koordinasi antara sesama anggota.

REFERENSI

<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf>

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

<https://www.neliti.com/id/publications/135138/pengenalan-wajah-menggunakan-algoritma-eigenface-dan-euclidean-distance>

<https://www.gramedia.com/literasi/perkalian-matriks/#!/history>

LAMPIRAN

Link Repository GitHub : <https://github.com/semitfinal-com/Algeo02-21080>

Link Bonus Video : <https://www.youtube.com/watch?v=yUdvujtZt-A>