

LAPORAN TUGAS BESAR 2

IF2123 Aljabar Linier dan Geometri
“Aplikasi Nilai Eigen dan EigenFace pada Pengenalan Wajah (Face Recognition)”

KELAS K01
Dosen : Dr. Judhi Santoso, M.Sc.



Disusul oleh :
Kelompok (KomukEigen)

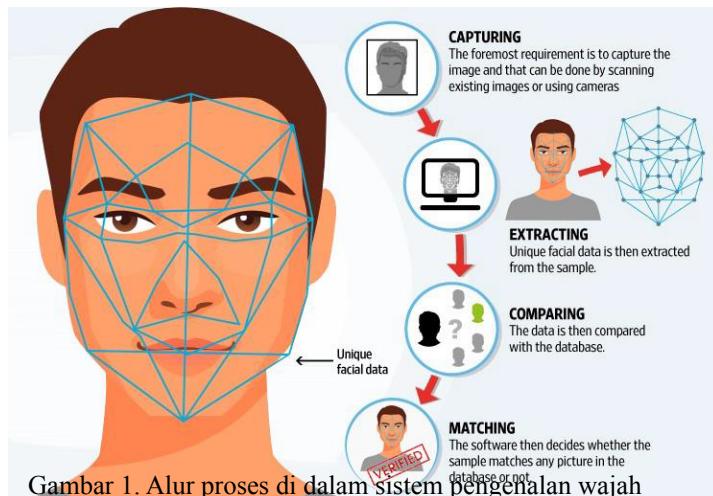
Anggota:
Irsyad Nurwidian Basuki (13521072)
Muhammad Zaydan A (13521104)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

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 Gambar 1.



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 pencocokkan. 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.

BAB II

Teori Singkat

2.1. Perkalian Matriks

Perkalian matriks adalah nilai pada matriks yang bisa dihasilkan dengan cara dikalikan-nya tiap baris dengan setiap kolom yang memiliki jumlah baris yang sama. Setiap anggota matriks ini nantinya akan dikalikan dengan anggota elemen matriks lainnya.

Perkalian matriks ini dilakukan sesuai urutan dan aturan yang berlaku pada perkalian bilangan matriks. Saat sedang menghitung nilai suatu matriks, berarti akan melihat adanya kolom dan juga baris. Kolom dan baris digunakan untuk menentukan maupun menghitung nilai matriks. Pada dasarnya kolom dan baris sangat diperlukan dalam penghitungan matriks.

Dalam matematika, perkalian matriks adalah suatu operasi biner dari dua matriks yang menghasilkan sebuah matriks. Agar dua matriks dapat dikalikan, banyaknya kolom pada matriks pertama harus sama dengan banyaknya baris pada matriks kedua. Matriks hasil perkalian keduanya, akan memiliki baris sebanyak baris matriks pertama, dan kolom sebanyak kolom matriks kedua. Perkalian matriks **A** dan **B** dinyatakan sebagai **AB**.

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

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

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

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

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

2.1. Nilai Eigen

Jika A adalah sebuah matriks $n \times n$, maka sebuah vektor taknol \mathbf{x} pada \mathbb{R}^n disebut vektor eigen (vektor karakteristik) dari A jika $A\mathbf{x}$ adalah sebuah kelipatan skalar dari \mathbf{x} ; jelasnya:

$$A\mathbf{x} = \lambda\mathbf{x}$$

untuk skalar sebarang λ . Skalar λ ini disebut nilai eigen (nilai karakteristik) dari A , dan \mathbf{x} disebut sebagai vektor eigen (vektor karakteristik) dari A yang terkait dengan λ .

Contoh:

Diberikan vektor $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ dan matriks $A = \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix}$.

$$A\mathbf{x} = \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 3\mathbf{x}$$

Maka, vektor $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ disebut vektor eigen dari matriks $A = \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix}$ yang terkait dengan nilai eigen $\lambda = 3$.

Untuk memperoleh nilai eigen dari sebuah matriks A berukuran $n \times n$, persamaan $A\mathbf{x} = \lambda\mathbf{x}$ dapat dituliskan kembali menjadi

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

Agar λ dapat menjadi nilai eigen, harus terdapat satu solusi taknol dari persamaan ini. Persamaan ini memiliki solusi taknol jika dan hanya jika

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

Menentukan nilai-nilai eigen

$$\lambda I - A = \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix} = \begin{bmatrix} \lambda - 3 & 0 \\ -8 & \lambda + 1 \end{bmatrix}$$

$$\det(\lambda I - A) = 0 \rightarrow \begin{vmatrix} \lambda - 3 & 0 \\ -8 & \lambda + 1 \end{vmatrix} = 0 \rightarrow (\lambda - 3)(\lambda + 1) = 0 \rightarrow \text{persamaan karakteristik}$$
$$\rightarrow \lambda_1 = 3 \text{ dan } \lambda_2 = -1$$

Jadi, nilai-nilai eigen dari matriks A adalah $\lambda = 3$ dan $\lambda = -1$.

2.2. Vektor Eigen

Menentukan vektor eigen

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

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

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

Jadi, $\begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}$ adalah basis untuk ruang eigen dengan $\lambda = 3$

Ruang eigen ditulis sebagai $E(3) = \{ \mathbf{x} = t \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}, t \in \mathbb{R} \}$

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

$$\begin{bmatrix} -4 & 0 \\ -8 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Selesaikan dengan eliminasi Gauss:

$$\begin{bmatrix} -4 & 0 & 0 \\ -8 & 0 & 0 \end{bmatrix} \xrightarrow{\text{R1}/(-4)} \begin{bmatrix} 1 & 0 & 0 \\ -8 & 0 & 0 \end{bmatrix} \xrightarrow{\text{R2} + 8\text{R1}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

\rightarrow Solusi: $x_1 = 0, x_2 = t, t \in \mathbb{R}$

Vektor-vektor eigen: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ t \end{bmatrix} = t \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow$ membentuk **ruang eigen (eigenspace)**

Jadi, $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ adalah basis untuk ruang eigen dengan $\lambda = -1$

Ruang eigen ditulis sebagai $E(-1) = \{ \mathbf{x} = t \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t \in \mathbb{R} \}$

2.3. EigenFace

Berikut merupakan langkah rinci dalam pembentukan eigenface.

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

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

2. Langkah kedua adalah ambil nilai tengah 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 \mathbb{C}

$$\mathbb{C} = \frac{1}{M} \sum_{n=1}^M \phi_n \phi_n^T = AA^T \quad (4)$$

$$L = A^T A \quad L = \phi_m^T \phi_n$$

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

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

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

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

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

BAB III

IMPLEMENTASI PROGRAM

Untuk mengimplementasi face recognition ini, kami telah membagi program menjadi beberapa modularitas sebagai berikut :

1. Main.py

```
def setdata(foldername):
    images = []
    for folder in os.listdir(foldername):
        for imgfile in os.listdir(foldername+folder):
            img = cv2.imread(os.path.join(foldername+folder, imgfile), cv2.IMREAD_GRAYSCALE)
            img = img/255
            img = cv2.resize(img,(256,256))
            images.append(img.reshape(-1,1))
            #vector face dibuat dari 256x256 jadi 65536 x M
            #dimasukan ke face vector space (images)
    return images
```

Fungsi **setdata** ini berfungsi untuk mengubah foto - foto yang ada di dataset menjadi matriks $N^2 \times 1$ dan disimpan di dalam matriks “images”

```
def average(arr):
    sum = [[0 for i in range(1)] for j in range(256*256)]
    a = []
    for i in range(ctr):
        sum = numpy.array(numpy.add(sum, arr[i]))
    mean = numpy.array(numpy.divide(sum, ctr))
    ...

    result = cv2.normalize(numpy.reshape(mean, (256,256)), dst=None, alpha=0, beta=255,norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    cv2.imwrite("kelaji.jpg", result)
    ...

    for j in range(ctr): #matrix a diisi normalized vector (tiap vector face - mean)
        a.append(numpy.array(numpy.subtract(arr[j], mean)))
    #hasil matrix a -> N^2 x M

    #buat hasilin average face

    return a #ngehasilin matrix isi normalized vectors (A)
```

Fungsi **average** ini berfungsi untuk mendapatkan vektor yang telah dinormalisasi, yaitu tiap vektor pada matriks dikurangi rata - rata dari vektor tersebut dan disimpan di matriks a.

```

def bigA(vectorM): #V
    biga = vectorM[0]
    for i in range(1, len(vectorM)):
        biga = numpy.hstack((biga, vectorM[i]))
    return biga

```

Fungsi **bigA** ini berfungsi untuk mengubah matriks 3D (tiga dimensi) menjadi matriks 2D(dua dimensi). Tujuan dibuat fungsi ini adalah karena fungsi setdata dan fungsi average menghasilkan matriks 3D. Oleh karena itu untuk mempermudah perhitungan berikutnya diubah menjadi matriks 2D.

```

def cofmatrix(normalvector): #V
    normal = normalvector
    transpose = numpy.transpose(normal)
    #cof = numpy.matmul(transpose, normal)
    cof = transpose @ normal
    return cof

```

Fungsi **cofmatrix** ini berfungsi untuk menghitung matriks kovarian. Pada fungsi ini digunakan perhitungan $AT \times A$ untuk menghasilkan ukuran matriks yang lebih kecil. Perlu diperhatikan bahwa perhitungan $AT \times A$ tetap menghasilkan nilai eigen yang sama.

```

def gramschmidt(A):
    R = numpy.zeros((A.shape[1], A.shape[1]))
    Q = numpy.zeros(A.shape)
    for k in range(0, A.shape[1]):
        R[k, k] = numpy.sqrt(numpy.dot(A[:, k], A[:, k]))
        Q[:, k] = A[:, k]/R[k, k]
        for j in range(k+1, A.shape[1]):
            R[k, j] = numpy.dot(Q[:, k], A[:, j])
            A[:, j] = A[:, j] - R[k, j]*Q[:, k]
    return Q, R

```

Fungsi **gram schmidt** ini merupakan fungsi untuk QR Decomposition, yang dimana mendekomposisi matriks menjadi matriks Q dan R dengan metode gram-schmidt

```
def qr_eigen(A):
    Ak = numpy.copy(A)
    n = Ak.shape[0]
    QQ = numpy.eye(n)
    for k in range(500):
        shift = Ak.item(n-1, n-1)
        shiftmult = shift * numpy.eye(n)

        Q, R = numpy.linalg.qr(numpy.subtract(Ak, shiftmult))

        Ak = numpy.add(R @ Q, shiftmult)
        QQ = QQ @ Q
    return numpy.diag(Ak), QQ
```

Fungsi **qr_eigen** merupakan fungsi yang digunakan untuk mendapatkan eigenvalue dan eigenvector. Penggunaannya adalah dengan memanfaatkan QR Decomposition. Nilai eigen dihasilkan dengan cara mengambil elemen diagonal dari matriks Ak dan vektor eigen dihasilkan dari hasil QQ.

```
def eigenface(matrix,ctr):
    w=[0 for i in range(ctr)]
    ui = []
    avg = bigA(average(matrix,ctr)) #V
    eigval, eigvec = qr_eigen(cofmatrix(bigA(average(matrix,ctr)))) #V

    #eigenfaces
    for i in range(ctr):
        ui.append((avg @ eigvec[:, i])/numpy.linalg.norm(avg @ eigvec[:,i]))

    #BUAT PRINT EIGENFACES
    ...
    for i in range(ctr):
        outfile = '%s.jpg' % ("eigens"+ str(i))
        result = cv2.normalize(numpy.reshape(ui[i][:], (256,256)), dst=None, alpha=0, beta=255,norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        cv2.imwrite(outfile, result)
    ...

    sum = [[0 for i in range(1)] for j in range(256*256)]
    for i in range(ctr):
        sum = numpy.array(numpy.add(sum, matrix[i]))
    mean = numpy.array(numpy.divide(sum, ctr))

    w = ui @ avg

    return ui, mean, numpy.array(w), avg
```

Fungsi **eigenface** ini merupakan fungsi yang digunakan untuk mencari eigenface serta mencari weight dari tiap eigenface tersebut. Pencarian eigenface dapat dilakukan dengan pengalian dari tiap normalisasi vektor dengan hasil vektor eigen yang telah diperoleh. Untuk mendapatkan

weight dari tiap eigenface dilakukan dengan pengalian tiap eigenface dengan tiap normalisasi vektor yang telah ada.

```
def eucdistance(m1,m2):
    selisih = m1 - m2
    eucDist = numpy.linalg.norm(selisih)
    return eucDist
```

Fungsi **eucdistance** ini merupakan fungsi untuk menghitung jarak euclidean dari dua matriks.

```
def mainprog(imginput,path):
    ctr = 0
    for folder in os.listdir(path):
        for imgfile in os.listdir(path+"/"+folder):
            ctr+=1

    ui, mean, w, avg = eigenface(setdata(path),ctr)

    #NYOCOKIN
    images = []
    print(imginput)
    img = cv2.imread(rf'{imginput}', cv2.IMREAD_GRAYSCALE)
    print(imginput)
    img = img / 255
    img = cv2.resize(img, (256, 256))
    images.append(img.reshape(-1,1))
    testimage = bigA(images)
    result = cv2.normalize(numpy.reshape(testimage, (256,256)), dst=None, alpha=0, beta=255,norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    # cv2.imwrite("YangdiTest.jpg", result)

    normalize = testimage - mean

    testWeigth = ui @ normalize

    euc = []

    for i in range(ctr):
        euc.append(eucdistance(testWeigth[:,0], w[:,i]))

    for i in range(ctr): #Mencari jarak euclidean terkecil
        if euc[i] == numpy.amin(euc):
            break

    # print("YANG DICARI GAMBAR KE ",i)
    x = setdata(path)

    result = cv2.normalize(numpy.reshape(bigA(x)[:,i], (256,256)), dst=None, alpha=0, beta=255,norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    cv2.imwrite("identified.jpg", result)
    imgresult=os.path.abspath("identified.jpg")
    return imgresult
```

Fungsi **mainprog** ini merupakan fungsi utama untuk menjalankan keseluruhan program. Fungsi ini memanggil fungsi untuk memproses dataset dan memproses test image yang ingin diuji.

Garis besar algoritma yang digunakan adalah dengan mereduksi dimensi serta penggunaan konsep aljabar linier untuk mengenali wajah. Dimulai dengan pemrosesan dataset yang besar lalu diolah dengan penggunaan matriks kovarian dan dekomposisi QR untuk mendapatkan nilai eigen & vektor eigen. Selanjutnya adalah pemrosesan eigenfaces yang telah dihasilkan serta

mendapatkan *weight* dari tiap eigenfaces. Untuk *face recognition* dapat dilakukan dengan perhitungan jarak euclidean.

2.gui.py

```
def start():
    global running
    if ((img1txt.get()=='') or (path1.get()=='')):
        return
    if not running:
        running = True
        timebefore=time.time()
        #ada prosedur eigen
        imgoutputmmmain=main.mainprog(img1txt.get(),path1.get())
        imgoutput.set(imgoutputmmmain)
        img_fotooutput=Image.open(imgoutput.get()).resize((500,500))
        img_fotooutput=ImageTk.PhotoImage(img_fotooutput)
        img_label2.configure(image=img_fotooutput)
        img_label2.image=img_fotooutput

        timeafter=time.time()

        duration=round((timeafter-timebefore),3)
        clock1.config(text=str(duration))

        running=False
        # print(imgoutput.get())
```

start() merupakan prosedur yang digunakan untuk menjalankan mainprogram dengan input yang ditentukan oleh user dan mengeluarkan output dari mainprog

```
def open():
    img1filename=filedialog.askopenfilename(initialdir=".\\Algeo02-21072",title="Choose file",filetypes=(("jpg files","*.jpg"),("all files","*.*")))

    if (img1filename):
        img1txt.set(img1filename)
        imginputtxt.set(img1filename)
        img1.set(img1filename)
        img_foto=Image.open(img1.get()).resize((500,500))
        img_foto=ImageTk.PhotoImage(img_foto)
        img_label.configure(image=img_foto)
        img_label.image=img_foto

        img1txtnlabel=img1txt.get()

        img1labeltxt.configure(text=img1txtnlabel)
        img1labeltxt.text=img1txtnlabel

        # img1input.configure(text=img1inputtxt.get())
        # img1input.text=img1inputtxt.get()
```

`open()` merupakan prosedur untuk melakukan insert image yang akan digunakan sebagai input mainprog

```
def opendir():
    path1name=filedialog.askdirectory(initialdir=".\\Algeo02-21072",title="Choose file")

    if (path1name):
        path1.set(path1name)

        temp=path1.get()

        path1label.configure(text= temp)
        path1label.text=temp

        # path1main.configure(text= temp)
        # path1main.text=temp
```

`opendir()` merupakan prosedur untuk memilih folder yang akan digunakan sebagai dataset

```
def get_time():
    timeVar=time.strftime("%I : %M : %S %p")
    clock.config(text=timeVar)
    clock.after(500,get_time)
```

prosedur clock/jam

3. guite.py

```
def realtime():
    # define a video capture object
    vid = cv2.VideoCapture(0)

    while(True):
        # Capture the video frame
        # by frame

        ret, frame = vid.read()
        # Display the resulting frame
        cv2.imshow('FR', frame)

        # the 'q' button is set as the
        # quitting button you may use any
        # desired button of your choice
        # if cv2.waitKey(1) & 0xFF == ord('q'):
        #     break
        if cv2.waitKey(1) & 0xFF == ord('q'):break
        if cv2.waitKey(10) & 0xFF == ord('a'):

            cv2.imwrite('riltme.jpg', frame)
            imgfile=r"C:\Users\ASUS\Documents\TBAlgeo2\Algeo02-21072\riltme.jpg"
            img1.set(imgfile)
            img1txt.set(imgfile)
            img_foto1=Image.open(img1.get()).resize((500,500))
            img_foto1=ImageTk.PhotoImage(img_foto1)
            img_label.configure(image=img_foto1)
            img_label.image=img_foto1

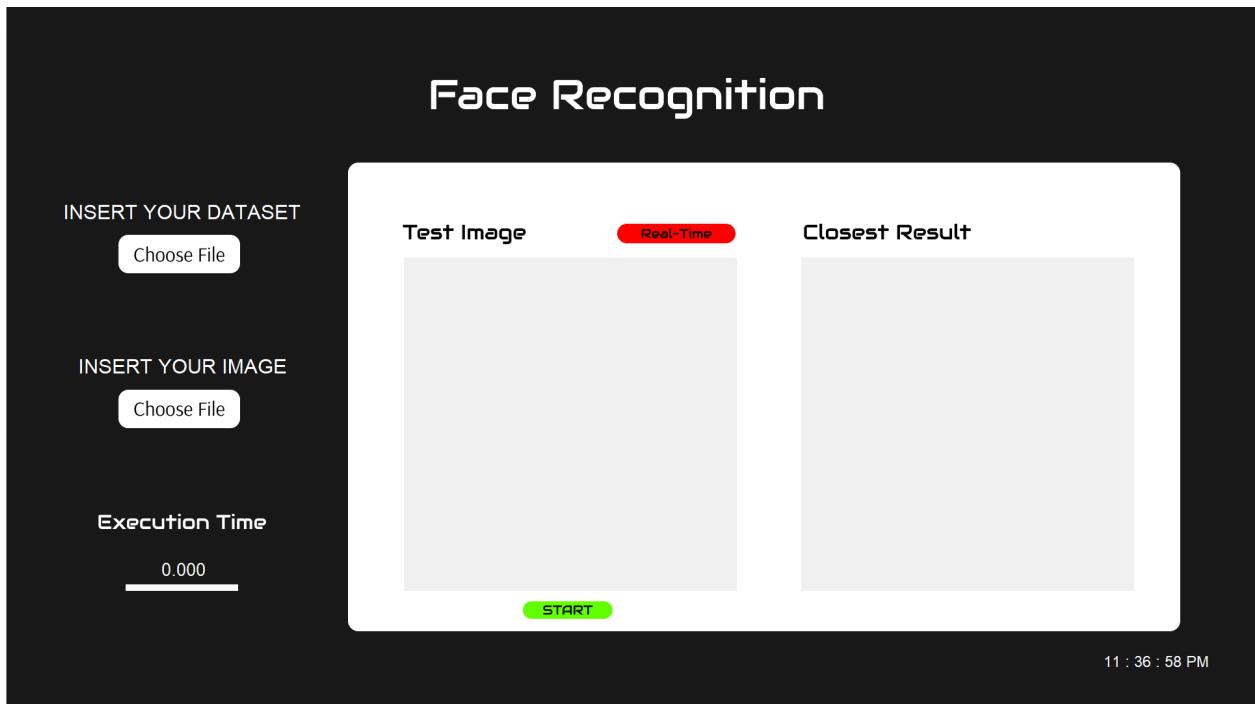
            img1labeltxttemp=img1txt.get()
            img1labeltxt.config(text=img1labeltxttemp)
            img1labeltxt.text=img1labeltxttemp
            start()
```

Prosedur untuk mengakses foto secara langsung melalui kamera device

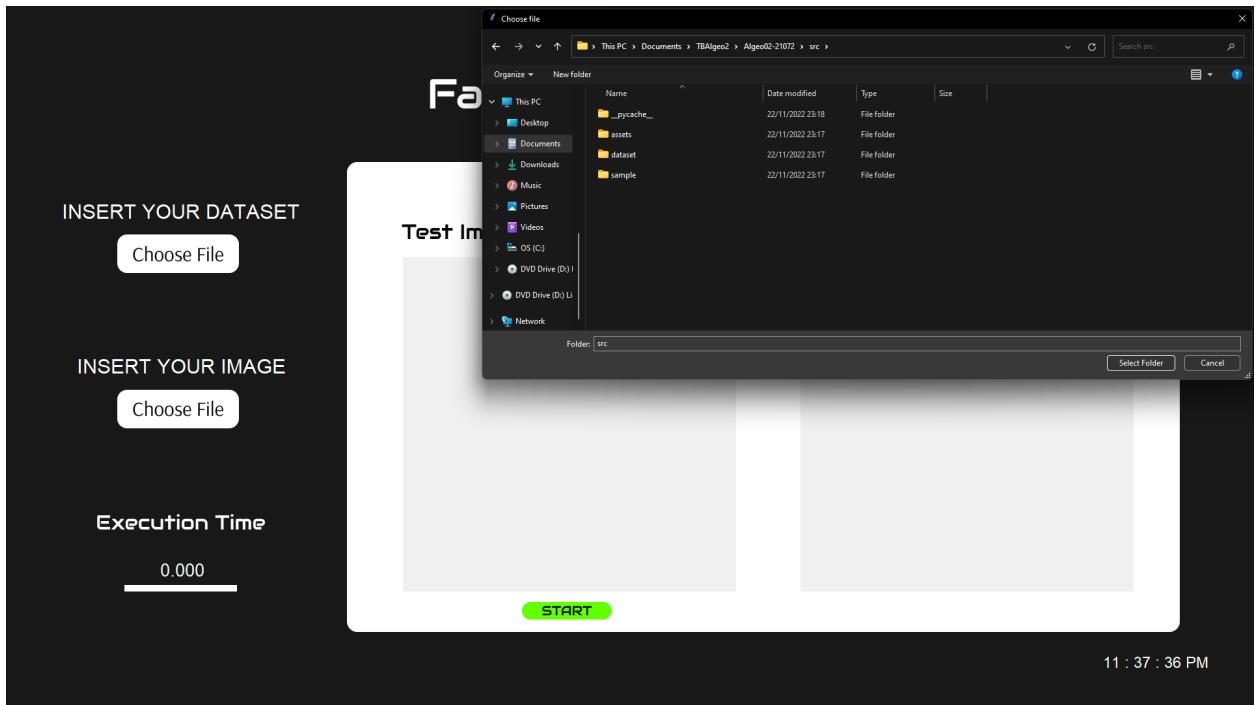
BAB IV

EKSPERIMENT

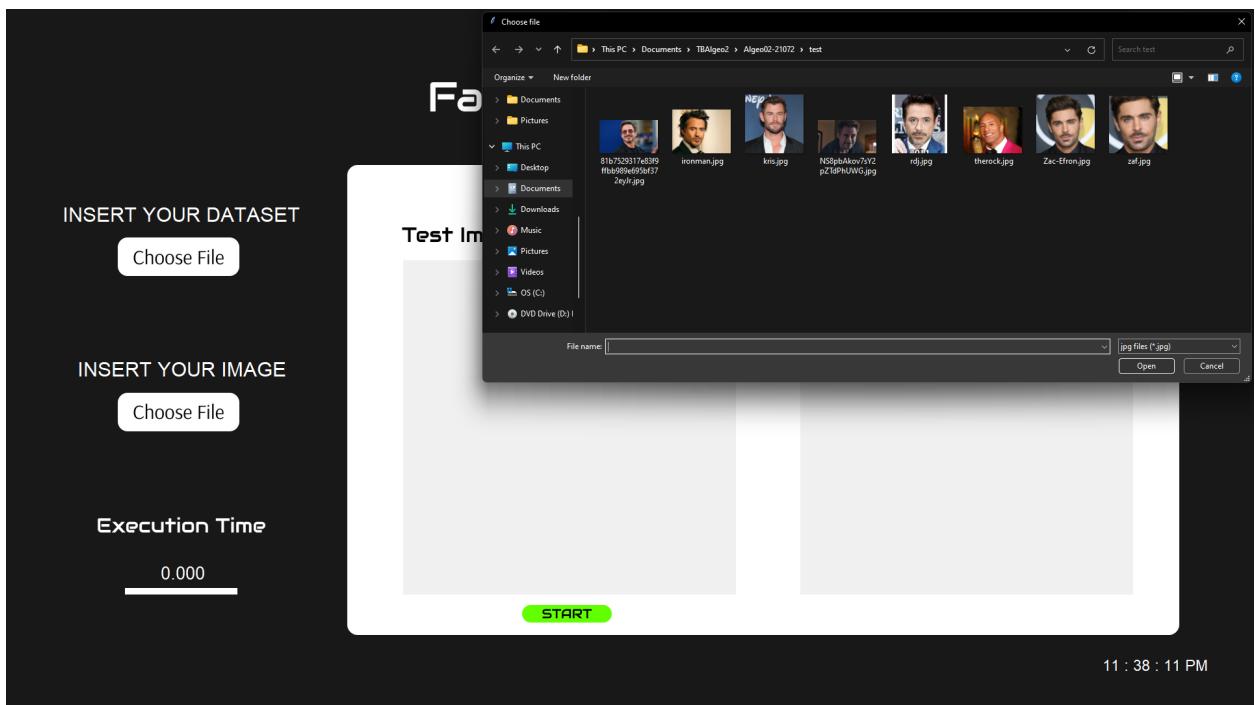
- Tampilan Awal



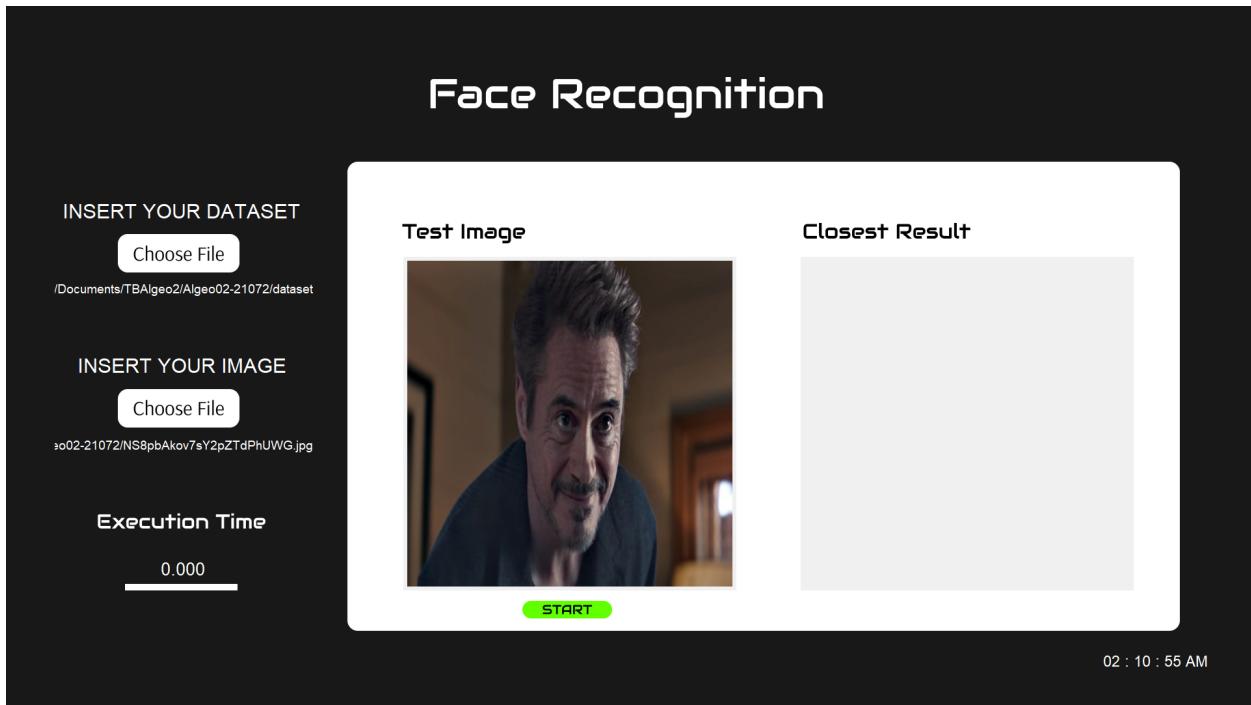
● Memilih Dataset



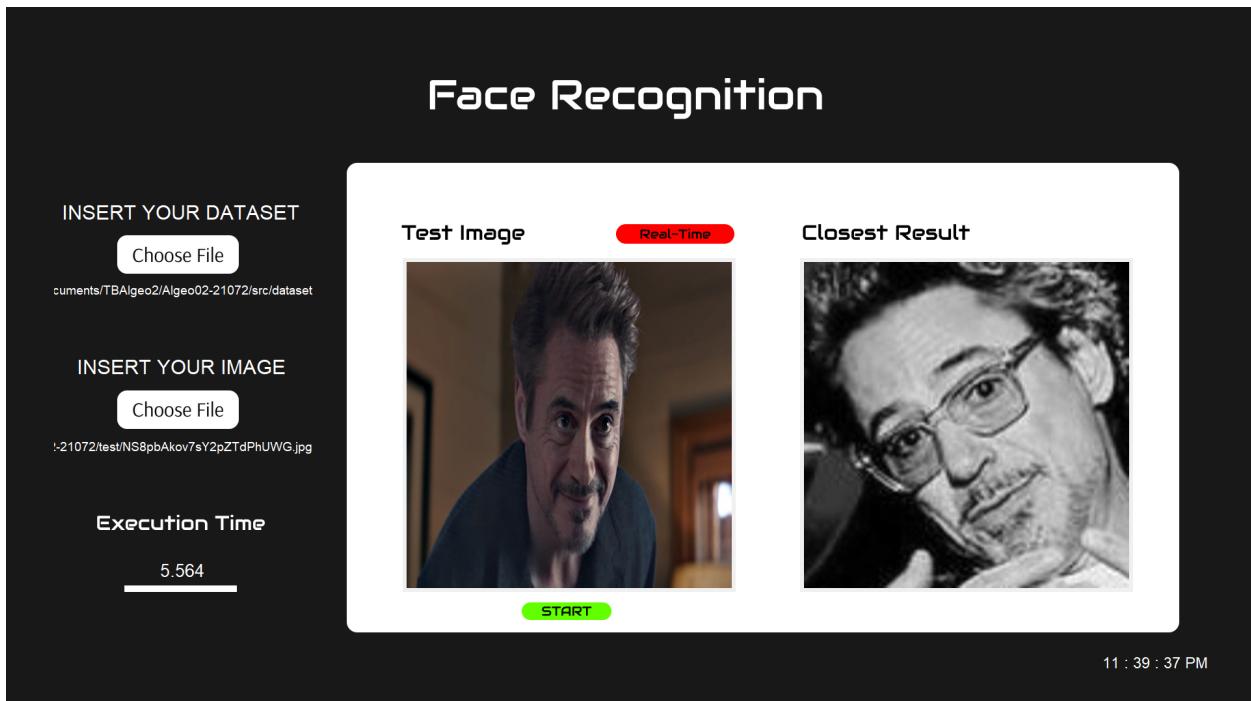
● Memilih Gambar



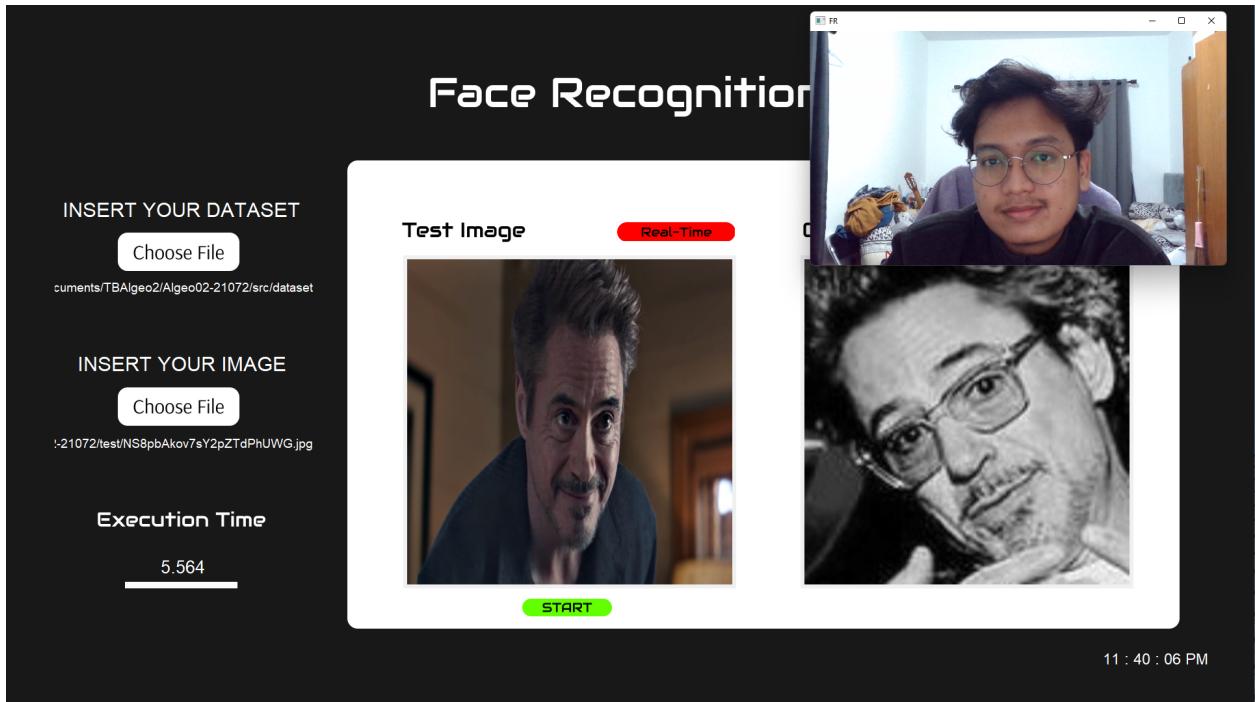
- Tampilan Sebelum Start (Face Recognition)



- Tampilan Sesudah Start (Face Recognition)



- Tampilan Real-Time



- **NOTE:**
 - Apabila user belum melakukan *Insert Dataset* atau *Insert Image*, menekan tombol START tidak akan terjadi apapun
 - Realtime clock di pojok kanan bawah

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

5.1. Kesimpulan

1. Kami berhasil membuat program Face Recognition dengan menggunakan metode eigenface dan dengan algoritma PCA (Principal Component Analysis).
2. Kami berhasil membuat UI program dengan menggunakan GUI python tkinter.
3. Kami berhasil mengimplementasikan perkalian matriks, nilai eigen, dan vektor eigen di program eigenface

5.2. Saran

Kedepannya program bisa dikembangkan lagi, apabila diberikan waktu tambahan. Contoh pengembangan yang bisa dilakukan adalah dengan memperbaiki user interface, untuk mempermudah dalam menggunakan program. Dengan UI yang baik, pengguna lebih mudah untuk melakukan input ke dalam program dan lebih mudah untuk membaca hasil perhitungan dari program. Pembuatan matriks eselon dan eselon tereduksi juga bisa diubah dari bentuk desimal ke bentuk pecahan, untuk meningkatkan keakuratan dan tampilan yang lebih bagus. Kami merasa waktu yang dibutuhkan untuk menjalankan program *face recognition* dapat lebih dipersingkat lagi.

5.3. Refleksi

Permasalahan time-management menjadi masalah utama bagi kami, karena kami harus membagi waktu dengan mata kuliah lain, kegiatan diluar kuliah, sehingga seringkali Tugas Besar ini terbengkalai. Masalah kedua adalah anggota yang menghilang dan tidak membantu mengerjakan tugas besar sama sekali. Masalah ketiga adalah perlunya eksplorasi yang sangat luas untuk mendapatkan konsep dari penggunaan eigenface untuk face recognition ini. Untuk itu diperlukan waktu luang yang lebih untuk bisa mengerjakan tugas besar ini.

REFERENSI

Wikipedia :

<https://en.wikipedia.org/wiki/Eigenface>

GeeksforGeeks :

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

ICT UoM (YouTube) :

<https://www.youtube.com/watch?v=61NuFIK5VdU&t=392s>

Nasir, Mahvish (YouTube) :

<https://www.youtube.com/watch?v=SaEmG4wcFfg&t=1046s>

LAMPIRAN

Link repository GitHub : <https://github.com/irsyadnb/Algeo02-21072>

Link video demo (YouTube) : <https://youtu.be/TWV4EaZedbc>

Link video bonus (YouTube) : <https://youtu.be/8Xf3RBdpNvM>