

LAPORAN TUGAS BESAR 2
IF2123 ALJABAR LINIER DAN GEOMETRI
APLIKASI NILAI EIGEN DAN EIGENFACE PADA
PENGENALAN WAJAH (*FACE RECOGNITION*)



Kelompok Seven11:

Michael Leon Putra Widhi (13521108)

Kenneth Dave Bahana (13521145)

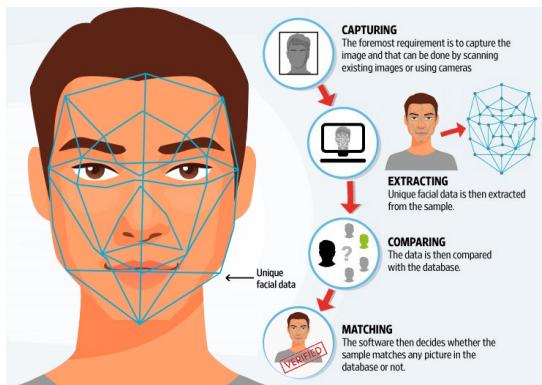
Nathan Tenka (13521172)

**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.



Gambar 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.

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.

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.

Kode program untuk ekstraksi fitur dapat dibaca pada artikel ini: *Feature extraction and similar image search with OpenCV for newbies*, pada laman:

<https://medium.com/machine-learning-world/feature-extraction-and-similar-image-search-with-opencv-for-newbies-3c59796bf774>

Nilai batas kemiripan citra test face dapat ditentukan oleh pembuat program melalui percobaan.

Berikut merupakan referensi pengenalan wajah dengan metode *eigenface*:

<https://jurnal.untan.ac.id/index.php/jcskommipa/article/download/9727/9500>

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

BAB II

TEORI SINGKAT

A. Perkalian Matriks

Ada 2 jenis perkalian matriks yang bisa dilakukan, yaitu perkalian matriks dengan skalar dan perkalian matriks dengan matriks lain. Perkalian matriks dengan skalar bisa dilakukan dengan mengalikan setiap elemen matriks dengan bilangan tersebut, sedangkan perkalian matriks dengan matriks lain hanya bisa dilakukan bila kedua matriks berukuran $m \times k$ dan $k \times n$. Matriks yang dihasilkan dengan perkalian kedua matriks tersebut akan berukuran $m \times n$. Tahapan perkalian matriks dengan matriks adalah sebagai berikut :

1. Misalkan matriks A dan B sebagai berikut :

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \dots & \dots & \dots & \dots \\ B_{m1} & B_{m2} & \cdots & B_{mn} \end{bmatrix}$$

Gambar 2.1.1. Matriks Sembarang A dan B

2. Maka matriks $C = AB$ didefinisikan sebagai berikut :

$$C = \begin{bmatrix} C_{11}C_{12}\dots\dots C_{1c} \\ C_{21}C_{22}\dots\dots C_{2c} \\ \dots\dots\dots \\ C_{a1}C_{a2}\dots\dots C_{ac} \end{bmatrix}$$

Gambar 2.1.2. Matriks C Hasil Perkalian A dan B

dengan

$$C_{xy} = A_{x1}B_{y1} + \dots + A_{xb}B_{by} = \sum_{k=1}^b A_{xk}B_{ky}$$

Gambar 2.1.3. Perhitungan Isi Matriks C

untuk $x = 1\dots a$ dan $y = 1\dots c$

B. Nilai dan Vektor Eigen

Nilai eigen (λ) dan vektor eigen (\mathbf{x}) adalah suatu skalar dan vektor tidak nol yang memenuhi persamaan

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

Gambar 2.2.1. Persamaan untuk Mencari Nilai Eigen

dengan A merupakan sebuah matriks berukuran $n \times n$. Nilai eigen dan vektor eigen dihitung sebagai berikut

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ I\mathbf{x} &= \lambda\mathbf{x} \quad (\text{kalikan kedua ruas dengan } I = \text{matriks identitas}) \\ A\mathbf{x} &= \lambda\mathbf{x} \\ (\lambda I - A)\mathbf{x} &= 0 \end{aligned}$$

Gambar 2.2.2. Cara Menghitung Nilai dan Vektor Eigen

Supaya persamaan tersebut memiliki solusi non-trivial, persamaan berikut harus terpenuhi

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

Gambar 2.2.3. Persamaan Karakteristik Matriks A

Persamaan di atas juga disebut persamaan karakteristik dari matriks A. Akar-akar dari persamaan tersebut adalah nilai eigen dari matriks A.

Setelah nilai-nilai eigen didapatkan, vektor eigen bisa didapat dengan menyelesaikan SPL

$$(\lambda I - A)\mathbf{x} = 0$$

Gambar 2.2.4. Persamaan untuk Mencari Vektor Eigen

untuk semua λ . Akan tetapi, perhitungan dengan cara di atas kurang efisien jika dilakukan di komputer, sehingga pada tugas ini digunakan metode-metode numerik seperti *QR Algorithm* dan *Rayleigh Quotient Iteration*.

QR Algorithm melakukan dekomposisi QR (dekomposisi matriks menjadi matriks ortogonal Q dan matriks segitiga atas R) pada matriks secara berulang-ulang hingga didapat matriks yang *similar* dengan matriks awal yang berbentuk matriks segitiga atas. Elemen diagonal pada matriks tersebut adalah nilai eigen matriks. Lalu, karena matriks kovarian bisa dipastikan simetris (karena merupakan hasil perkalian matriks dengan transposenya), vektor

eigennya pasti ortogonal dan hasil perkalian semua matriks Q dari iterasi akan menghasilkan vektor-vektor eigen yang dicari. Secara spesifik, *QR Algorithm* yang digunakan untuk tugas ini adalah variasi dengan menggunakan *Wilkinson Shift* dan deflasi untuk mempercepat konvergensi dan perhitungan. Berikut adalah algoritma umumnya dilansir dari

<https://mathoverflow.net/questions/258847/solved-how-to-retrieve-eigenvectors-from-qr-algorithm-that-applies-shifts-and-d> dan

<https://pi.math.cornell.edu/~web6140/TopTenAlgorithms/QRalgorithm.html>

```

set  $H_0 := H$ 
for  $m = n, \dots, 2$  do
     $k = 0$ 
    repeat
         $k = k + 1$ 
         $\sigma_k = \text{Wilkinson}(H_{k-1})$ 
         $H_{k-1} - \sigma_k I =: Q_k R_k \quad (*)$ 
         $H_k = R_k Q_k + \sigma_k I$ 
        until  $|h_{m,m-1}^{(k)}| < \epsilon$ 
         $\lambda_m = h_{m,m}^{(k)}$ 
         $H^{(0)} = H_{1:(m-1),1:(m-1)}^{(k)}$ 
    end for

```

Gambar 2.2.5. *QR Algorithm* dengan *Wilkinson Shift* dan deflasi

$$\mu_k = c - \frac{\text{sign}(\delta)b^2}{|\delta| + \sqrt{\delta^2 + b^2}}$$

Gambar 2.2.6. Perhitungan *Wilkinson Shift*

Untuk dekomposisi QR-nya sendiri digunakan metode rotasi Givens. Rotasi Givens adalah matriks yang jika dikalikan akan meng-nol-kan komponen bawah dari sebuah vektor berukuran 2x1. Matriks sudah diubah dulu menjadi bentuk tridiagonal sehingga rotasi Givens menjadi efisien sebab hanya elemen subdiagonal yang perlu di-nol-kan.

Rayleigh Quotient adalah sebuah bilangan yang merupakan aproksimasi terhadap nilai eigen sebuah matriks berdasarkan vektor eigen \mathbf{x} . Persamaannya adalah sebagai berikut

$$r(x) = \frac{x^T A x}{x^T x}.$$

Gambar 2.2.7. Perhitungan Rayleigh Quotient

Rayleigh Quotient ini bisa digunakan untuk menghitung nilai dan vektor eigen dengan relatif cepat jika digabungkan dengan metode lain bernama *Inverse Power Iteration*. *Inverse Power Iteration* sendiri adalah algoritma untuk mencari vektor eigen berdasarkan nilai eigen. Algoritmanya adalah sebagai berikut

Algorithm 27.2. Inverse Iteration

```
 $v^{(0)}$  = some vector with  $\|v^{(0)}\| = 1$ 
for  $k = 1, 2, \dots$ 
    Solve  $(A - \mu I)w = v^{(k-1)}$  for  $w$            apply  $(A - \mu I)^{-1}$ 
     $v^{(k)} = w/\|w\|$                            normalize
     $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$            Rayleigh quotient
```

Gambar 2.2.8. Algoritma Inverse Iteration

Karena Rayleigh Quotient mengaproksimasi nilai eigen berdasarkan vektor eigen, dan *Inverse Iteration* mengaproksimasi vektor eigen berdasarkan nilai eigen, bisa didapat algoritma yang memiliki konvergensi kubik dengan menggabungkan kedua metode tersebut. Algoritma ini disebut *Rayleigh Quotient Iteration*.

Algorithm 27.3. Rayleigh Quotient Iteration

```
 $v^{(0)}$  = some vector with  $\|v^{(0)}\| = 1$ 
 $\lambda^{(0)} = (v^{(0)})^T A v^{(0)}$  = corresponding Rayleigh quotient
for  $k = 1, 2, \dots$ 
    Solve  $(A - \lambda^{(k-1)} I)w = v^{(k-1)}$  for  $w$      apply  $(A - \lambda^{(k-1)} I)^{-1}$ 
     $v^{(k)} = w/\|w\|$                            normalize
     $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$            Rayleigh quotient
```

Gambar 2.2.9. Algoritma Rayleigh Quotient Iteration

Untuk tugas ini, karena diketahui vektor eigen yang dihasilkan pasti ortogonal, algoritma di atas bisa diulang-ulang dengan menggunakan vektor v yang tegak lurus dengan semua vektor eigen sebelumnya hingga didapat semua nilai dan vektor eigen matriks.

C. EigenFace

Nilai eigen dan vektor eigen bisa digunakan untuk melakukan pengenalan wajah. Algoritmanya didasarkan pada PCA (*Principal Component Analysis*), yaitu teknik

mengurangi dimensionalitas persoalan. Pada algoritma *eigenface*, pengurangan dimensionalitas dilakukan memakai nilai eigen dan vektor eigen serta memproyeksikan sample data pada ruang *feature* yang kecil. Langkah algoritma Eigenface adalah sebagai berikut :

Algoritma training

1. Misalkan ada m gambar dalam *training set* dengan masing-masing gambar berukuran $n \times n$. Pertama-tama ubah setiap matriks gambar menjadi vektor berukuran $n^2 \times 1$.
2. Hitung vektor rata-rata dari semua vektor gambar tadi, lalu kurangi setiap vektor gambar dengan vektor rata-rata. Masukkan semua vektor yang sudah dikurangi ke dalam matriks sehingga didapat matriks A berukuran $n^2 \times m$.
3. Hitung matriks kovarian dengan mengalikan A dengan transpose-nya. Namun, supaya matriks kovarian tidak berukuran terlalu besar, bisa digunakan matriks kovarian hasil perkalian A transpose dengan A, sehingga didapat matriks berukuran $m \times m$ (dengan asumsi $m \ll n^2$).

$$\boxed{Cov = A^T A}$$

Gambar 2.3.1. Perhitungan Matriks Kovarian

4. Hitung nilai dan vektor eigen dari matriks kovarian tersebut. Lalu, untuk mendapat eigenface, kalikan matriks A dengan vektor eigen yang didapat.
5. Ambil k eigenface yang berkorespondensi dengan k nilai eigen terbesar ($k < m$).
6. Ambil vektor-vektor gambar yang sudah dikurangi vektor rata-rata dan nyatakan setiap vektor sebagai kombinasi linier dari seluruh eigenface yang diambil.
7. Simpan koefisien-koefisien kombinasi linier tersebut.

Algoritma testing

1. Proses gambar supaya memiliki ukuran yang sama dengan gambar dalam *training set* (jika ukurannya berbeda).
2. Ambil vektor gambar uji dan kurangi dengan vektor rata-rata yang sudah didapat sebelumnya.

3. Proyeksikan vektor gambar uji ke *eigenspace* supaya didapat koefisien kombinasi linier gambar uji terhadap setiap eigenface.
4. Hitung jarak euclidean koefisien gambar uji dengan tiap koefisien gambar *training set*. Gambar dengan jarak minimum dan kurang dari batas yang ditentukan adalah gambar yang sesuai.

BAB III

IMPLEMENTASI PROGRAM

Program ini dibuat dengan memanfaatkan library numpy (untuk operasi-operasi matriks), opencv (untuk ekstraksi fitur gambar), glob (untuk pengambilan gambar dari file), math (untuk operasi matematik), time (untuk mendapat waktu eksekusi), dan tkinter (untuk GUI).

Pemakaian program dimulai dengan pertama memilih *directory* untuk folder dataset *training*. Gambar pada dataset maupun gambar uji akan diubah menjadi berukuran 256 x 256 lebih dulu. Lalu, dataset tersebut akan diproses sesuai dengan algoritma eigenface yang sudah dijelaskan pada subbab 2.3. Pemrosesan training set hanya dilakukan saat pengguna memasukkan dataset baru. Setelah pengguna memasukkan folder dataset, pengguna bisa memasukkan gambar uji untuk mendapat gambar yang paling dekat dengan gambar uji tersebut. Algoritma nilai dan vektor eigen yang digunakan bisa diubah melalui kode. Secara *default* akan digunakan *QR Algorithm* dengan fungsi *QR* dari NumPy, karena metode tersebut adalah yang tercepat.

Struktur program terdiri dari interface.py sebagai program utama (mengandung GUI dan memanggil fungsi-fungsi yang lain), Eigenface.py yang berisi program untuk menghitung Eigenface dan menguji gambar wajah, dan camRecord.py untuk mengambil dataset dari kamera. Untuk folder gambar uji yang digunakan bisa menggunakan dari folder test maupun dari directory lain. Lalu, Berdasarkan hasil uji coba, didapat batas jarak minimum yang dapat ditoleransi adalah setengah dari jarak euclidean rata-rata. Jadi gambar dengan jarak minimum yang lebih besar dari batas tersebut dianggap tidak ada di dataset.

Berikut adalah detail mengenai struktur program yang kami buat :

a. interface.py

Nama Fungsi	Spesifikasi Fungsi
stopWebcam()	Menghentikan pembacaan wajah dari kamera.
main_webcams()	Menjalankan fitur pengenalan wajah melalui kamera terhadap dataset yang sudah dimasukkan.
capture()	Melakukan screenshot wajah dari kamera setiap 10 detik sekali.

reupdateResult(mulai,selesai)	Mengganti tampilan waktu eksekusi dan hasil gambar terdekat saat pembacaan dari kamera. Menerima parameter waktu mulai dan waktu selesai eksekusi fungsi.
reupdate()	Mengubah tampilan bagian test image sehingga bisa menampilkan video.
camerafunc()	Mengambil dataset wajah dari kamera.
insimg()	Menerima gambar uji dan melakukan pengenalan wajah terhadap dataset dengan masukan dari GUI dan mengubah waktu eksekusi serta tampilan gambar hasil. Menerima parameter
dataimg()	Menerima dataset melalui GUI dan memprosesnya untuk mendapat eigenface dan variabel-variabel lain yang diperlukan.

b. Eigenface.py

Jenis Fungsi	Nama Fungsi	Spesifikasi Fungsi
Fungsi dasar vektor	vectorLength(v)	Menghasilkan panjang vektor v. Menerima parameter sebuah vektor v.
Fungsi pendukung algoritma eigendecomposition	HouseHolder(vec)	Menghasilkan matriks refleksi Householder berdasarkan vektor vec. Menerima parameter vektor.
	Tridiagonalize(mtrx)	Mengubah matriks mtrx menjadi matriks tridiagonal. Menerima parameter matriks numerik.
	GivensRotation(a,b)	Mencari matriks rotasi Givens berukuran 2x2 berdasarkan vektor $[a \ b]^T$. Menerima dua bilangan a dan b.
	QRDecompTridiag(mtrx)	Melakukan dekomposisi QR pada matriks tridiagonal memakai rotasi Givens. Menerima matriks tridiagonal/Hessenberg atas.
	WilkinsonShift(a,b,c)	Menghitung nilai Wilkinson

		Shift berdasarkan submatriks 2x2 bawah kanan matriks. Menerima bilangan a ($M[n-1][n-1]$), b($M[n-1][n]$ sekaligus $M[n][n-1]$), dan c($M[n][n]$).
Algoritma perhitungan nilai dan vektor eigen	QREigenSendiri(mtrx)	Menghitung nilai dan vektor eigen matriks simetris mtrx memakai <i>QR Algorithm</i> dengan implementasi dekomposisi QR sendiri. Menerima parameter sebuah matriks simetris.
	QREigenBuiltIn(mtrx)	Menghitung nilai dan vektor eigen matriks simetris mtrx memakai <i>QR Algorithm</i> dengan dekomposisi QR dari Numpy. Menerima parameter sebuah matriks simetris.
	rayleigh_iteration(mtrx)	Menghitung nilai dan vektor eigen matriks simetris mtrx memakai <i>Rayleigh Quotient Iteration</i> . Menerima parameter sebuah matriks simetris.
Fungsi algoritma eigenface	InputFace(dir)	Menerima input dataset dari directory dir dan mengembalikan matriks vektor baris tiap gambar dataset (yang sudah diresize) dan matriks gambar awal yang masih berwarna. Menerima parameter path file dari dataset.
	MeanFace(imgVectorMatrix)	Mengembalikan vektor rata-rata berdasarkan matriks vektor imgVectorMatrix. Menerima parameter matriks vektor baris.
	EigenFace(imgVectorMatrix, method)	Mengembalikan eigenface, matriks koefisien gambar dataset terhadap eigenface, dan vektor rata-rata. Metode perhitungan nilai dan vektor eigen bisa diubah berdasarkan parameter method. Menerima matriks

		vektor baris dan string berisi metode eigendecomposition yang diinginkan.
	RecognizeFace(dir, eigenFace, coefTrain, mean, initImage)	Melakukan pengenalan wajah dari gambar uji yang ada di directory dir. Menuliskan gambar terdekat ke file yang sudah ditentukan. Algoritma sesuai dengan bab 2. Menerima parameter path gambar uji, matriks eigenface, matriks koefisien gambar dataset, vektor rata-rata, dan matriks gambar awal yang berwarna.

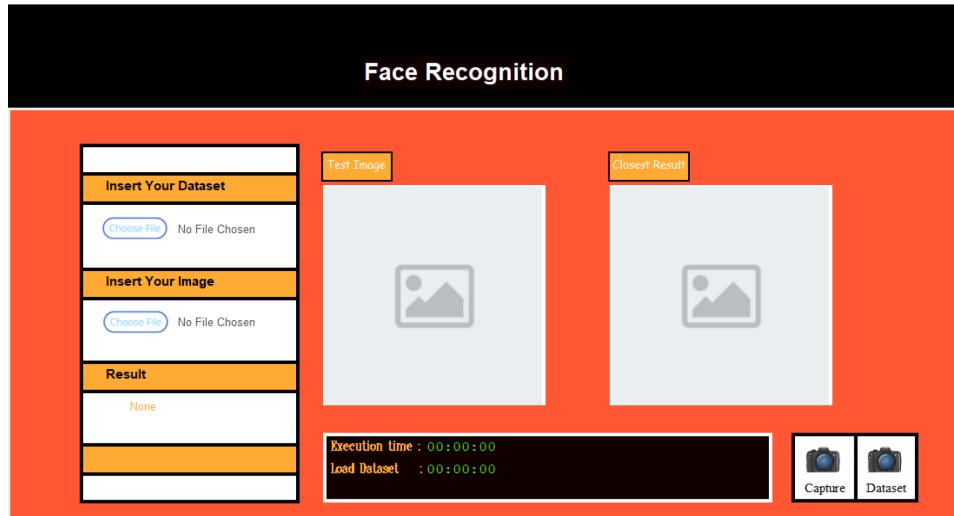
c. **camRecord.py**

Nama Fungsi	Spesifikasi Fungsi
main_cam()	Mengambil 30 gambar dari kamera untuk digunakan sebagai dataset.

BAB IV

EKSPERIMENT STUDI KASUS

A. Tampilan Awal GUI



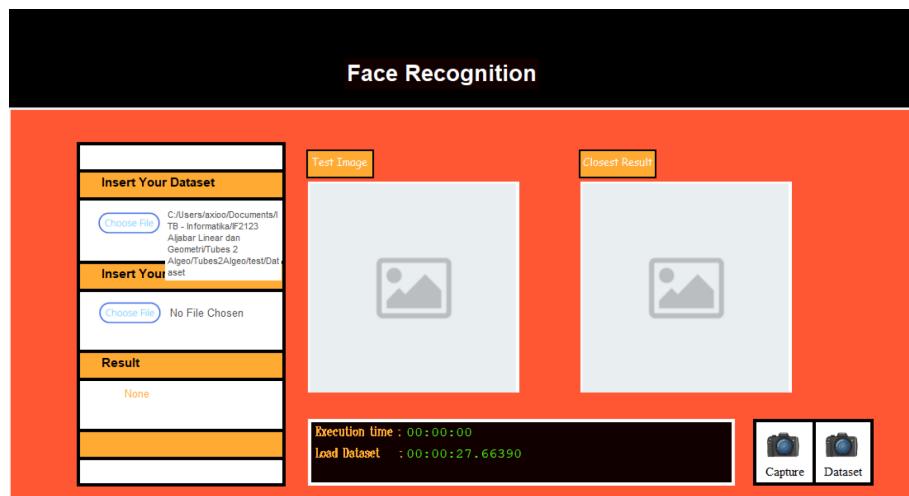
Gambar 4.1.1. Tampilan Awal Program

B. Pengujian Gambar dengan Algoritma Eigenface

1. Dataset : test/Dataset 1

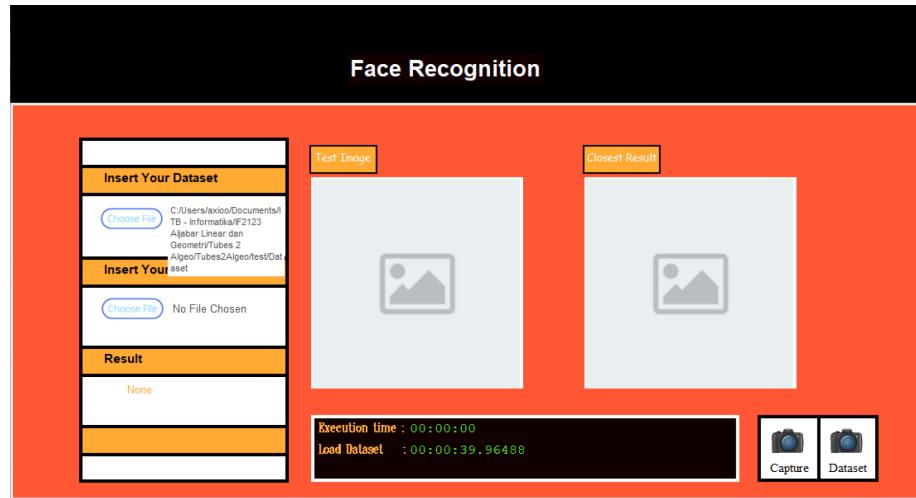
Deskripsi : 144 gambar, Ukuran konstan 302 x 320

a. Pengambilan Gambar dan pemrosesan Dataset



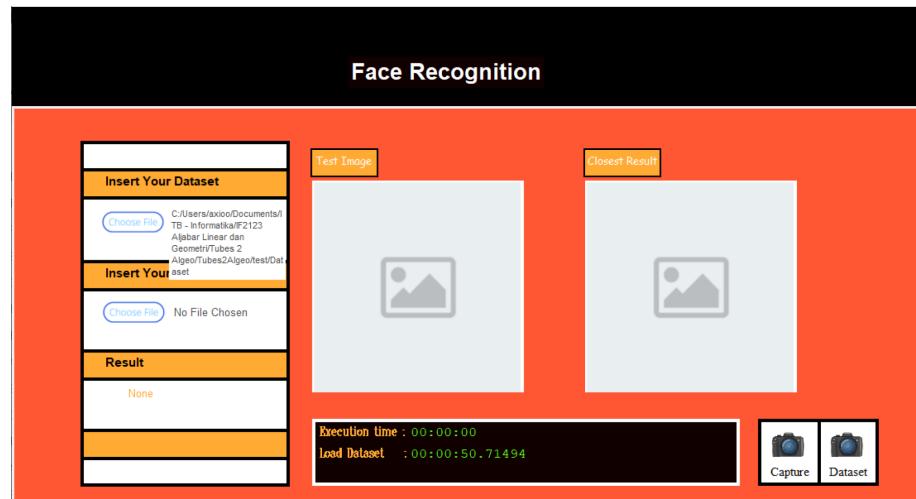
Gambar 4.2.1. Hasil Pemrosesan Dataset dengan metode QR hasil *Built-in*

Diperoleh waktu load dataset 27.664 sekon



Gambar 4.2.2. Hasil Pemrosesan Dataset dengan metode QR buatan sendiri

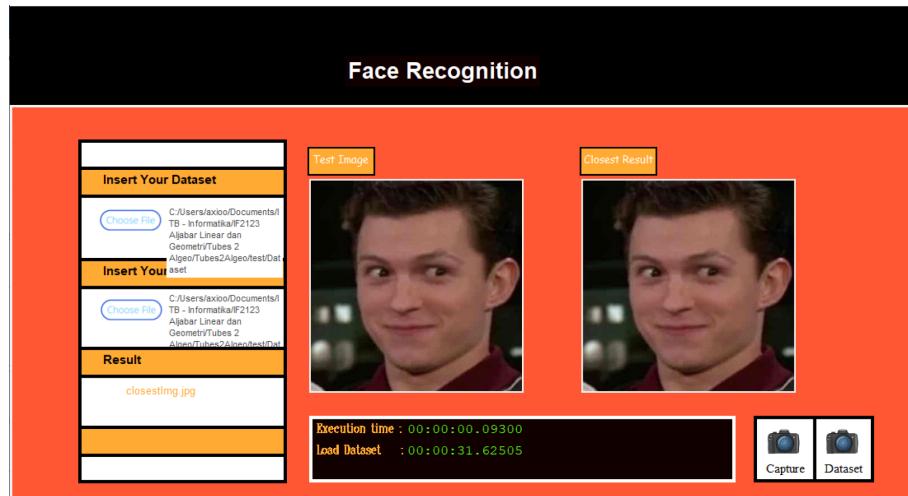
Diperoleh waktu load dataset 39.965 sekon



Gambar 4.2.3. Hasil Pemrosesan Dataset dengan metode Rayleigh

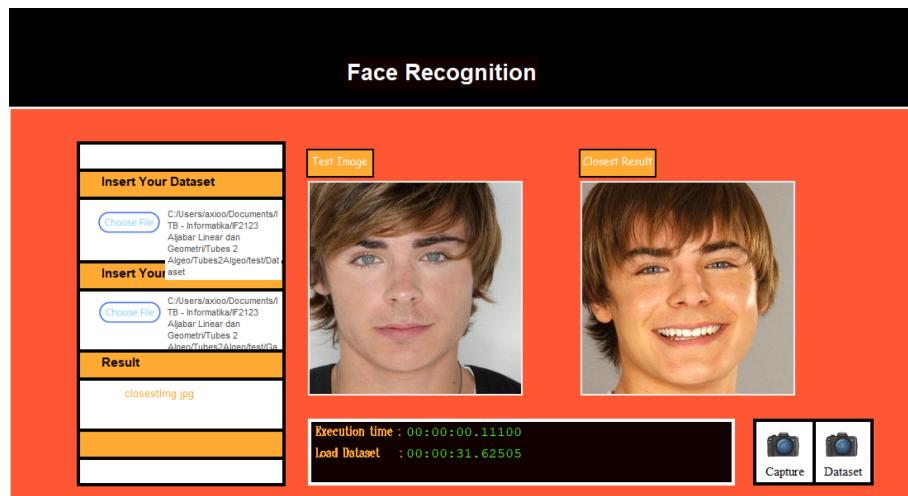
Diperoleh waktu load dataset 50.715 sekon

b. Gambar di dalam Dataset



Gambar 4.2.4. Hasil Uji 1 - Gambar di dalam Dataset

c. Gambar di luar Dataset

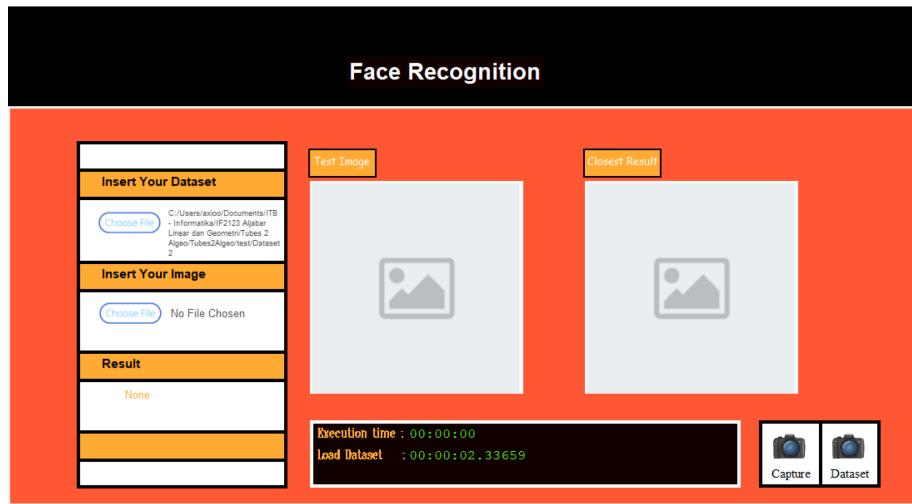


Gambar 4.2.5. Hasil Uji 2 - Gambar di luar Dataset

2. Dataset : test/Dataset 2

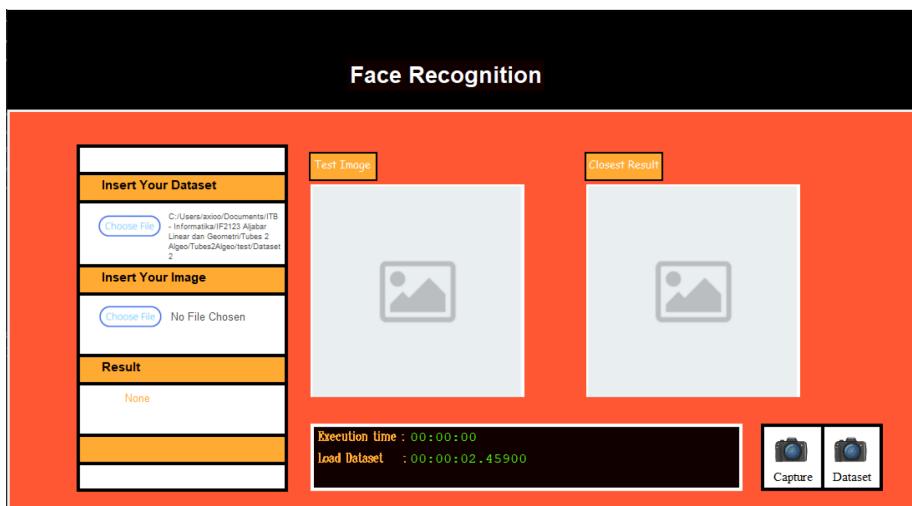
Deskripsi : 25 gambar, Ukuran konstan 256 x 256

a. Pengambilan Gambar dan pemrosesan Dataset



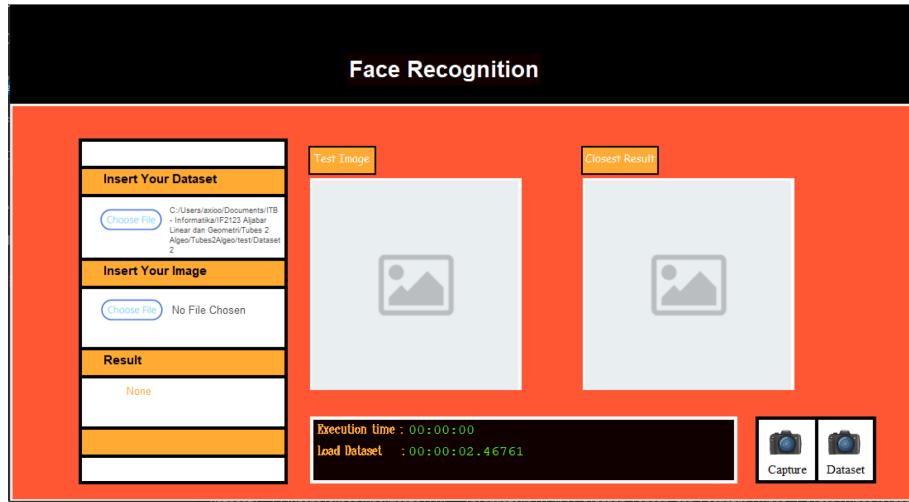
Gambar 4.2.6. Hasil Pemrosesan Dataset dengan metode QR hasil *Built-in*

Diperoleh waktu load dataset 2.336 sekon



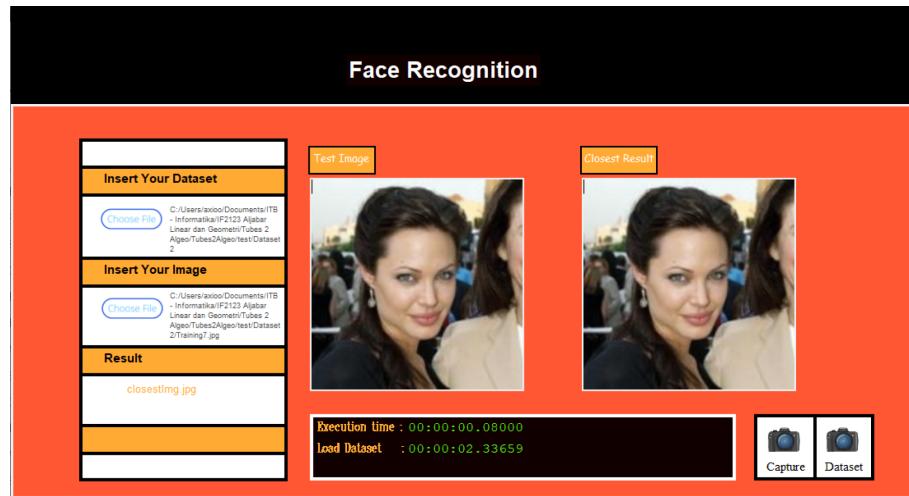
Gambar 4.2.7. Hasil Pemrosesan Dataset dengan metode QR buatan sendiri

Diperoleh waktu load dataset 2.459 sekon



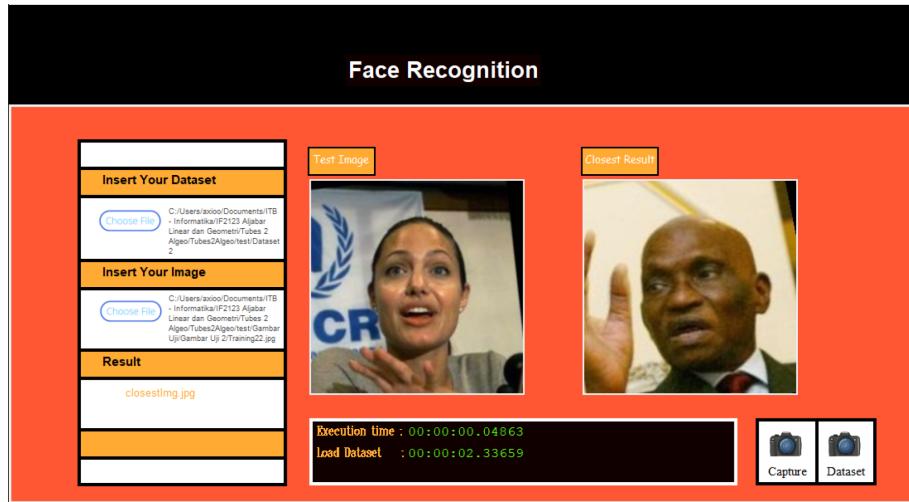
Gambar 4.2.8. Hasil Pemrosesan Dataset dengan metode Rayleigh
Diperoleh waktu load dataset 2.467 sekon

b. Gambar di dalam Dataset



Gambar 4.2.9. Hasil Uji 1 - Gambar di dalam Dataset

c. Gambar di luar Dataset

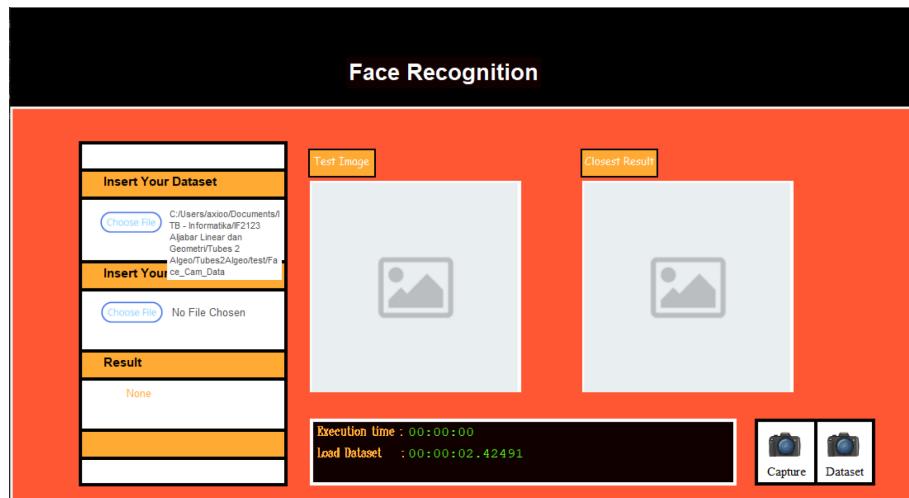


Gambar 4.2.10. Hasil Uji 2 - Gambar di luar Dataset

3. Dataset : test/Face_Cam_Data

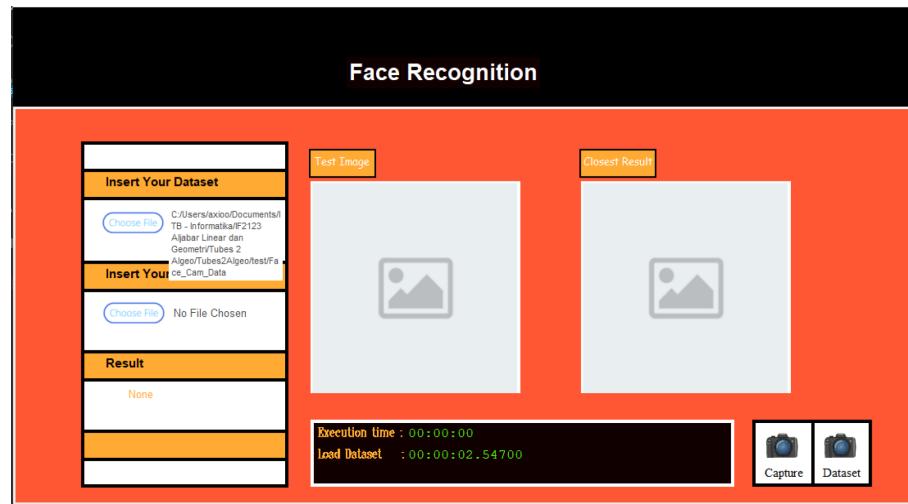
Deskripsi : 30 gambar hasil pengambilan kamera secara langsung, Ukuran konstan 256 x 256

a. Pengambilan Gambar dan pemrosesan Dataset

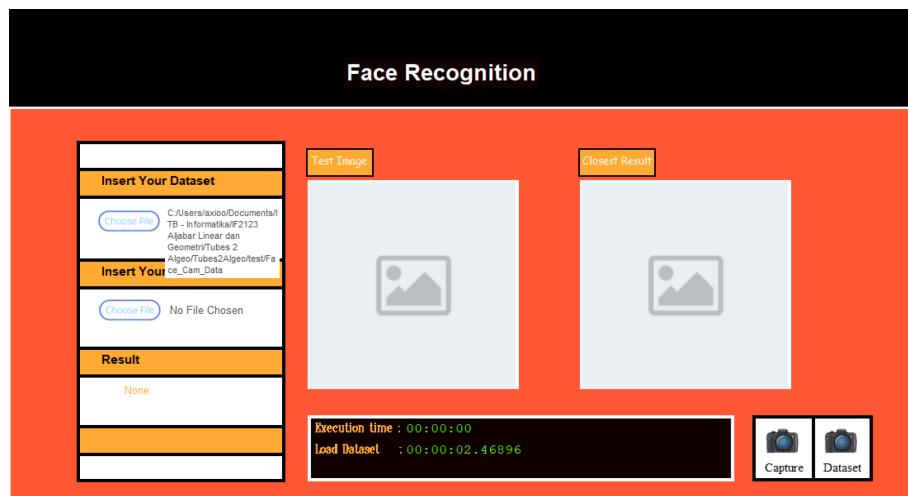


Gambar 4.2.11. Hasil Pemrosesan Dataset dengan metode QR hasil *Built-in*

Diperoleh waktu load dataset 2.429 sekon



Gambar 4.2.12. Hasil Pemrosesan Dataset dengan metode QR buatan sendiri
Diperoleh waktu load dataset 2.547 sekon



Gambar 4.2.13. Hasil Pemrosesan Dataset dengan metode Rayleigh
Diperoleh waktu load dataset 2.469 sekon

b. Gambar di dalam Dataset



Gambar 4.2.14. Hasil Uji 1 - Gambar di dalam Dataset

c. Gambar di luar Dataset



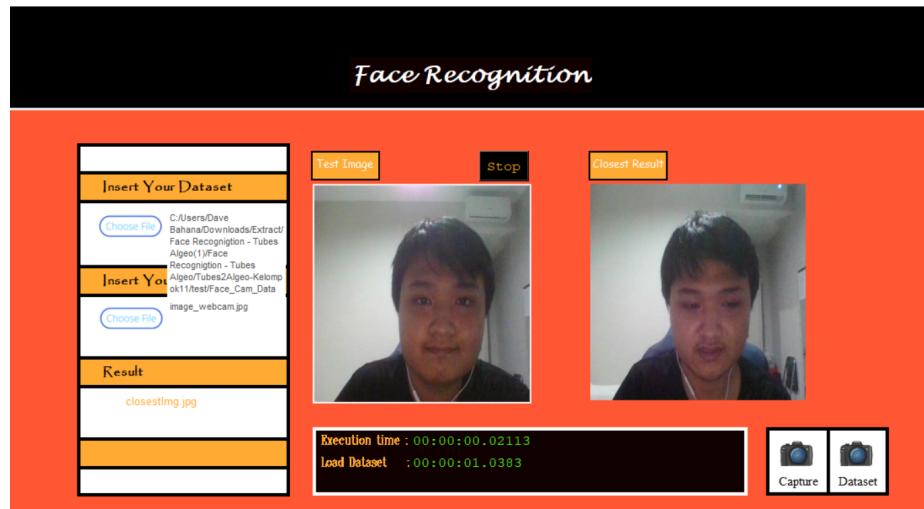
Gambar 4.2.15. Hasil Uji 2 - Gambar di luar Dataset

C. Pengujian Gambar Secara *Realtime* [Bonus]

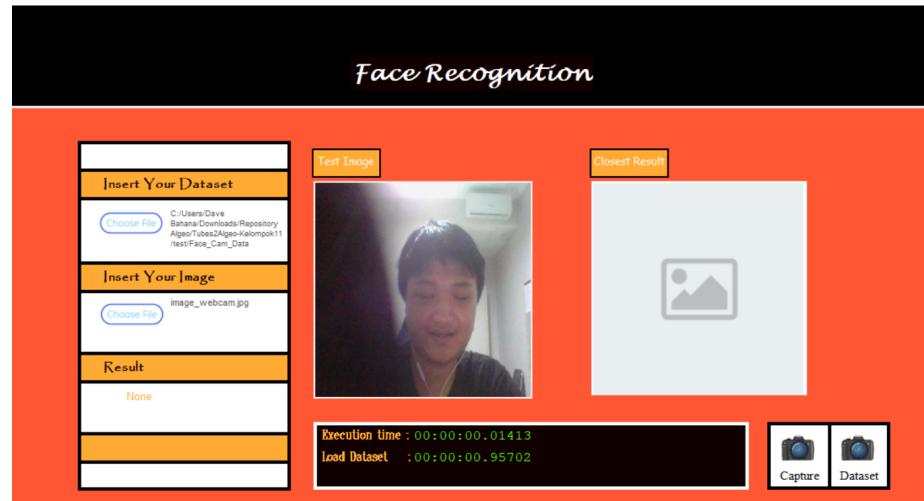
1. Dataset : test/Face_Cam_Data

Deskripsi : 30 gambar hasil pengambilan kamera secara langsung, Ukuran konstan 256 x

256



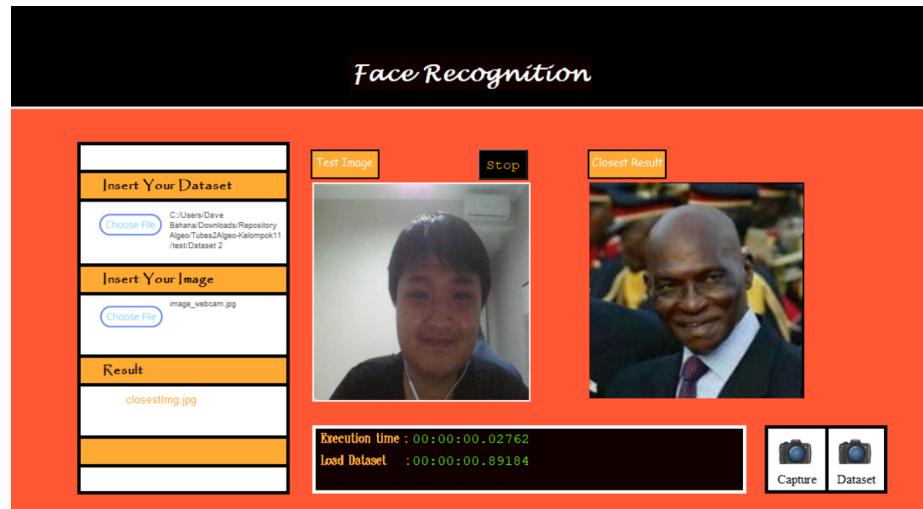
Gambar 4.3.1. Hasil Pengambilan Gambar 1, Terdapat gambar yang sesuai dibawah nilai jarak euclidean yang ditoleransi sehingga tercetak gambar dengan *closest result*



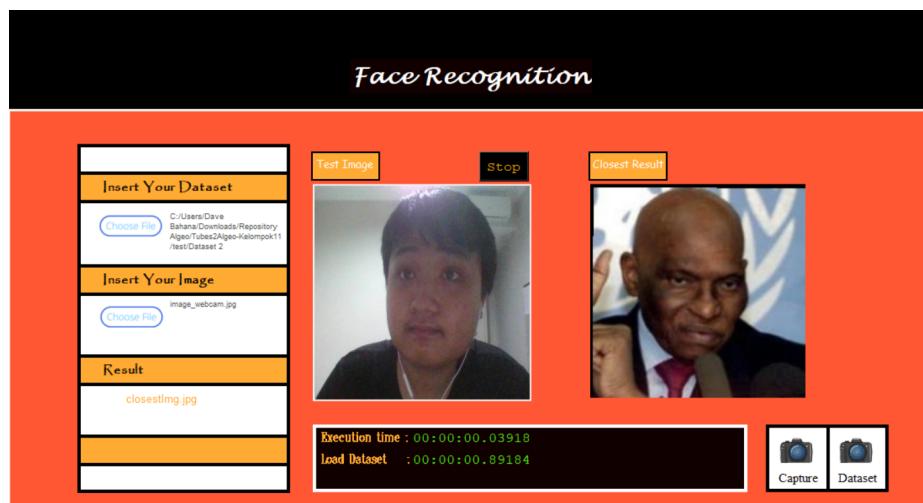
Gambar 4.3.2. Hasil Pengambilan Gambar 2, Tidak terdapat gambar yang sesuai dibawah nilai jarak euclidean yang ditoleransi sehingga tidak tercetak gambar dengan *closest result*

2. Dataset : test/Dataset 2

Deskripsi : 25 gambar, Ukuran konstan 256 x 256



Gambar 4.3.3. Hasil Pengambilan Gambar 1. Terdapat gambar yang sesuai dibawah nilai jarak euclidean yang ditoleransi sehingga tercetak gambar dengan *closest result* dengan menunjukkan kemiripan terdekat yang dimiliki dataset



Gambar 4.3.4. Hasil Pengambilan Gambar 2. Terdapat gambar yang sesuai dibawah nilai jarak euclidean yang ditoleransi sehingga tercetak gambar dengan *closest result* dengan menunjukkan kemiripan terdekat yang dimiliki dataset

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

A. Kesimpulan

Kesimpulan yang bisa ditarik dari tugas besar kedua Aljabar Linear dan Geometri ini adalah sebagai berikut :

1. Aplikasi yang menerapkan metode eigenface ini dapat digunakan untuk mengenal suatu wajah yang diuji namun keterbatasan dari metode yang bergantung pada faktor posisi wajah, cahaya, dan jarak muka terhadap kamera menyebabkan akurasi yang masih kurang baik dalam menemukan hasil wajah yang sesuai (Terbukti melalui hasil yang diperoleh pada Gambar 4.2.10)
2. Library Numpy Python dapat membantu penyelesaian fungsi - fungsi yang dibutuhkan dalam penggeraan *face recognition* ini.
3. Hasil eksekusi dan akurasi dari hasil pengenalan wajah bervariasi dikarenakan faktor variasi dari gambar yang diuji, dataset, dan pengaruh performansi serta *cache* berubah-ubah dari setiap komputer atau laptop yang digunakan berbeda - beda.

Secara umum program yang kami buat berjalan dengan baik. Akan tetapi, masih terdapat beberapa kekurangan dalam eksekusi sehingga masih perlu adanya perbaikan dan optimasi untuk menghasilkan program yang lebih baik.

B. Saran

Metode eigenface sendiri walaupun bisa digunakan untuk mencocokkan wajah, akurasinya masih cukup buruk apabila lingkungan tidak bersifat tetap. Hal tersebut dikarenakan kondisi kamera yang harus memadai, pencahayaan yang tetap, dan beberapa elemen yang harus bersifat konstan. Komponen utama yang diperhatikan dalam *face recognition* dengan metode eigenface adalah ukuran muka, jarak muka dari kamera, dan *lighting* dari kamera sebelum diujinya bentuk muka yang sesuai dengan orang pada gambar yang diuji.

C. Refleksi

Dari proses penggerjaan tugas besar ini, kami mendapatkan beberapa pelajaran dan pengalaman berharga yang kami dapatkan. Dimulai dari mempelajari metode eigenface serta belajar pembuatan GUI dengan bahasa TkInter, perencanaan pembagian tugas serta diskusi hasil akhir. Namun, beberapa keterbatasan ilmu mengenai penggunaan TkInter sendiri menyebabkan proses yang cukup sulit dan panjang untuk menggabungkan hasil algoritma *face recognition* dengan tampilan aplikasi dengan semaksimal mungkin.

BAB VI

DAFTAR REFERENSI

1. <https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/>
2. <https://byjus.com/math/matrix-multiplication/#algorithm>
3. <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf>
4. <https://pi.math.cornell.edu/~web6140/TopTenAlgorithms/QRalgorithm.html>
5. <https://mathoverflow.net/questions/258847/solved-how-to-retrieve-eigenvectors-from-qr-algorithm-that-applies-shifts-and-d>
6. <https://www.cs.cmu.edu/afs/cs/academic/class/15859n-f16/Handouts/TrefethenBau/RayleighQuotient-27.pdf>

LAMPIRAN

Link Repository : <https://github.com/Nat10k/Algeo02-21108>

Link Demo Video : <https://youtu.be/JfPGgpc7rM8>