République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Benyoucef Benkheda- Alger1

Master Ingénierie des Systèmes Informatiques Intelligents (ISII)

Mini-Projet de module traitement d'image numérique

Réalisation d'un système biométrique d'identification d'individus par l'iris

---

Réalisé par Groupe 11 :

- **BOUHADADOU Asma.**
- **ALLOUCHE Kenza.**
- **OUBARA Mouna.**

Table des matières

Liste de figure

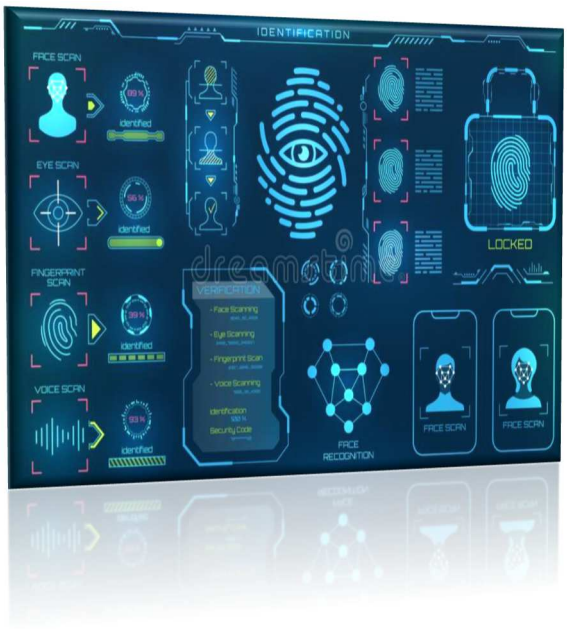
1 Introduction :

1.1 Introduction générale sur les systèmes biométriques :

Un système biométrique est essentiel un système de reconnaissance de formes qui fonctionne en acquérant des données biométriques à partir d'un individuel, extrayant un ensemble de caractéristiques à partir des données acquises, et comparant ces caractéristiques contre la signature dans la base de données. Selon le contexte d'application, un système biométrique peut fonctionner en mode de vérification ou mode d'identification :

- **Vérification :** le système valide l'identité d'une personne en comparant les données biométriques capturées à sa propre base de données. Dans un tel système, un individu qui désire être identifié réclame une identité, habituellement par l'intermédiaire d'un PIN (numéro d'identification personnelle), d'un nom d'utilisateur, ..., et le système conduit une comparaison d'un - a - un pour déterminer si la réclamation est vraie ou faux (est- ce que ces données biométriques appartiennent à tel ?)
- **Identification :** le système identifie un individu en recherchant les signatures (Template) de tous les utilisateurs dans la base de données. Par conséquent, le système conduit plusieurs des comparaisons pour établir l'identité d'un individu (ou échoue si le sujet n'est pas inscrit dans la base de données de système) sans devoir soumis réclamer une identité.

En général, tous les systèmes biométriques partagent le même schéma de fonctionnement.



1.2 Objectif :

L'objectif ce mini projet est de mettre en place un système de reconnaissance de l'iris pour identifier les personnes de manière unique et d'appliquer les bases étudiées en traitement d'image.

L'outil de développement utilisé est numpy, pil, matplotlib (pour les plots), Open CV (juste pour les détecteurs sift SIFT et déscripteur SIFT) en Python et d'autres bibliothèques python dépendantes ainsi que le logiciel utilisé anaconda, google collab, spyder.


Pour effectuer l'expérimentation, un ensemble de données Casia v1 contenant des images de l'œil humain prises sous différents angles.

2 Etat de l'art

2.1 Introduction

Il existe plusieurs techniques biométriques utilisées dans plusieurs applications et secteurs, et qui exploitent diverses informations biométriques à savoir : l'iris, le visage, la main, l'empreinte digitale, la voix, la signature …etc. Parmi ces différentes techniques on s'intéresse au système biométrique de détection par l'iris.

2.2 Définition de système biométriques de détection de l'individu par l'iris :

La reconnaissance de l'iris est une technologie plus récente puisqu'elle ne s'est véritablement développée que dans les années 80, principalement grâce aux travaux de J. Daugman. Après l'avoir localisé, on prend des photos en noir et blanc, on utilise ensuite des coordonnées polaires et on cherche les transformées en ondelettes, pour avoir finalement un code représentatif de l'iris. Et on utilise la distance de Hamming comme mesure de similarité, ou d'autres procédés.

La reconnaissance par iris est très utilisée dans les applications d'identification et de vérification, car il est hautement distinctif et unique, sa forme est stable et il est protégé et très robuste, toutefois les équipements d'acquisition coûtent chères. Cette technologie de l'iris est abordée en détails dans la suite du manuscript [1].

2.3 L'œil et l'iris

L'iris placé derrière la cornée de l'œil est un diaphragme variable percé d'un trou circulaire, la pupille (diamètre de 2,5 à 4,5 mm), régie par un sphincter et par un dilatateur formé de fibres musculaires antagonistes, lisses, rayonnantes et circulaires. La pupille s'agrandit quand les fibres musculaires sympathiques se contractent ; elle se rétrécit quand les fibres circulaires parasympathiques agissent. Cette ouverture de la pupille permet de modifier la quantité de lumière qui pénètre dans l'œil pour éviter l'aveuglement en plein soleil ou capter le peu de rayons lumineux la nuit

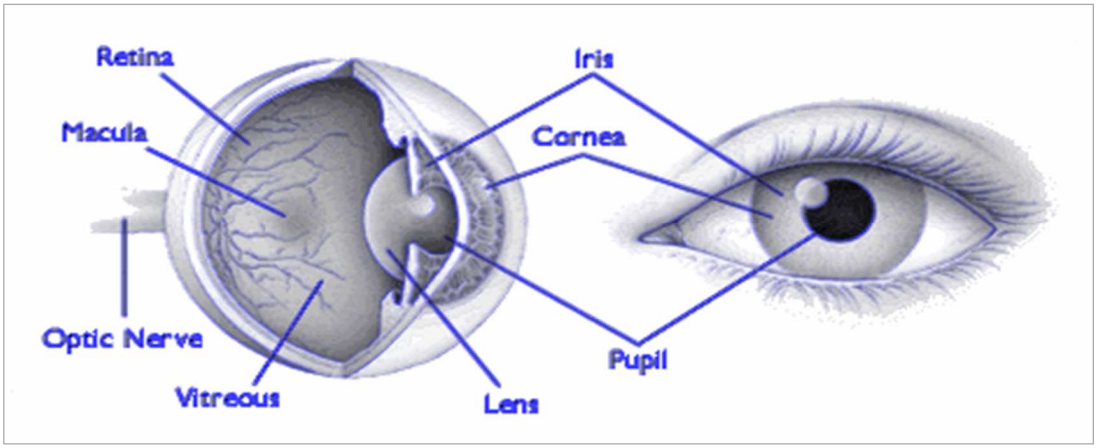


Figure 1 architecture de l'œil

2.4 Les avantages de système de détection de l'individu par l'iris :

- La structure est fixe à partir de l'âge d'un an environ et reste constante dans le temps
- Les variations des valeurs d'intensité du niveau de gris distinguent deux individus, la différence existe entre des jumeaux identiques et même entre l'œil gauche et l'œil droit d'une même personne
- Les caractéristiques ne sont pas génétiquement déterminées
- La précision s'est avérée beaucoup plus élevée par rapport à d'autres types de systèmes biométriques tels que les empreintes digitales, les empreintes de main et les empreintes vocales

3 Implémentation et réalisation :

3.1 Les phases de réalisation de système

Un tel système peut se décomposer en trois unités principales :

- Une unité d'acquisition des images de la base de données Casia v1.
- Une unité de prétraitement des données :
  - Amélioration de contraste et égalisation de l'histogrammes
  - Lissage de l'image
  - Segmentation de l'image
  - Les operations morphologiques
- Une unité d'extraction et comparaison des informations avec celles stockées préalablement lors de l'enrôlement.

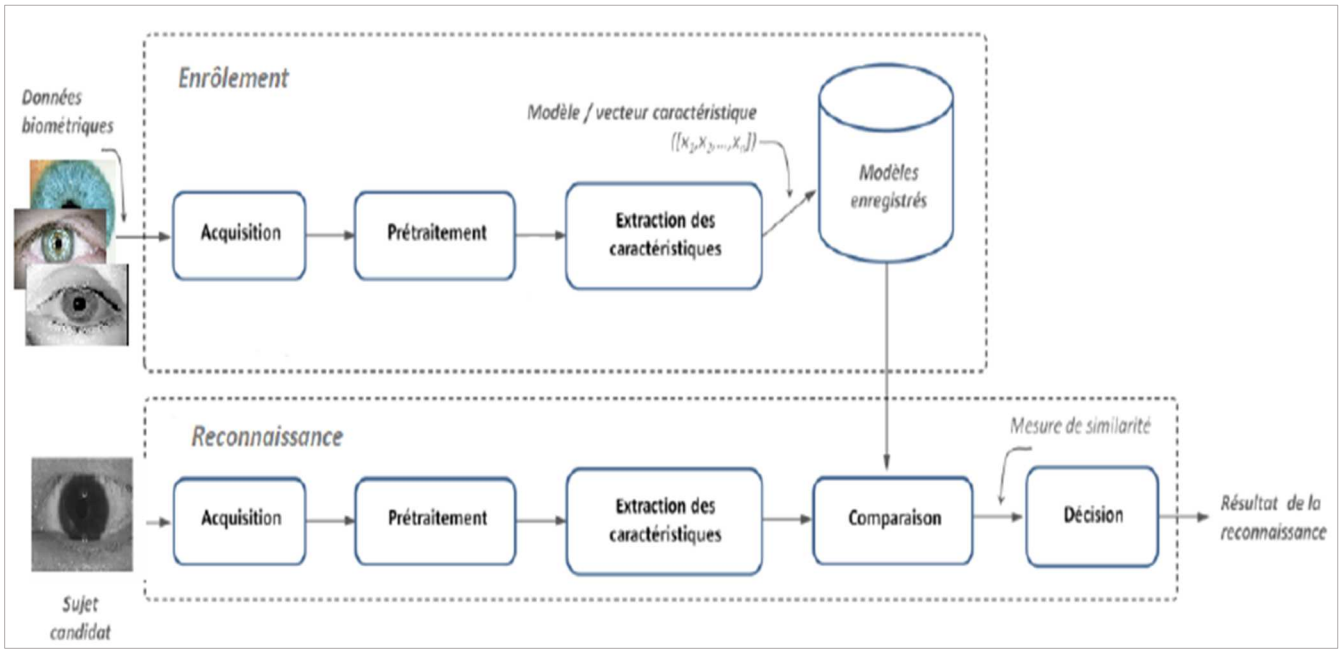


Figure 2 les phases de réalisation de système de détection par l'iris

3.2 Organisation de la Base de données CASIA-IrisV1 :

La base de données CASIA Iris V1 cette base de données est considérée comme étant très propre, Dans ce présent travail nous avons utilisé la base de données des images CASIA - IrisV1, sa répartition en classes est composée principalement par les images Iris des personnes asiatiques.

Chaque classe de l'Iris est composée de 7 échantillons du même œil. Nous avons créé 108 dossiers, le nom de chaque dossier est unique, et désigne une classe qui correspond à une personne spécifiée, les images de chaque classe sont renommées par un code décimal représente une certaine propriété utile liée à l'image, tel que l'adresse des images, en spécifiant la session, Vont de 1 à 3 dans la première session, et de 1 à 4 dans la deuxième session.

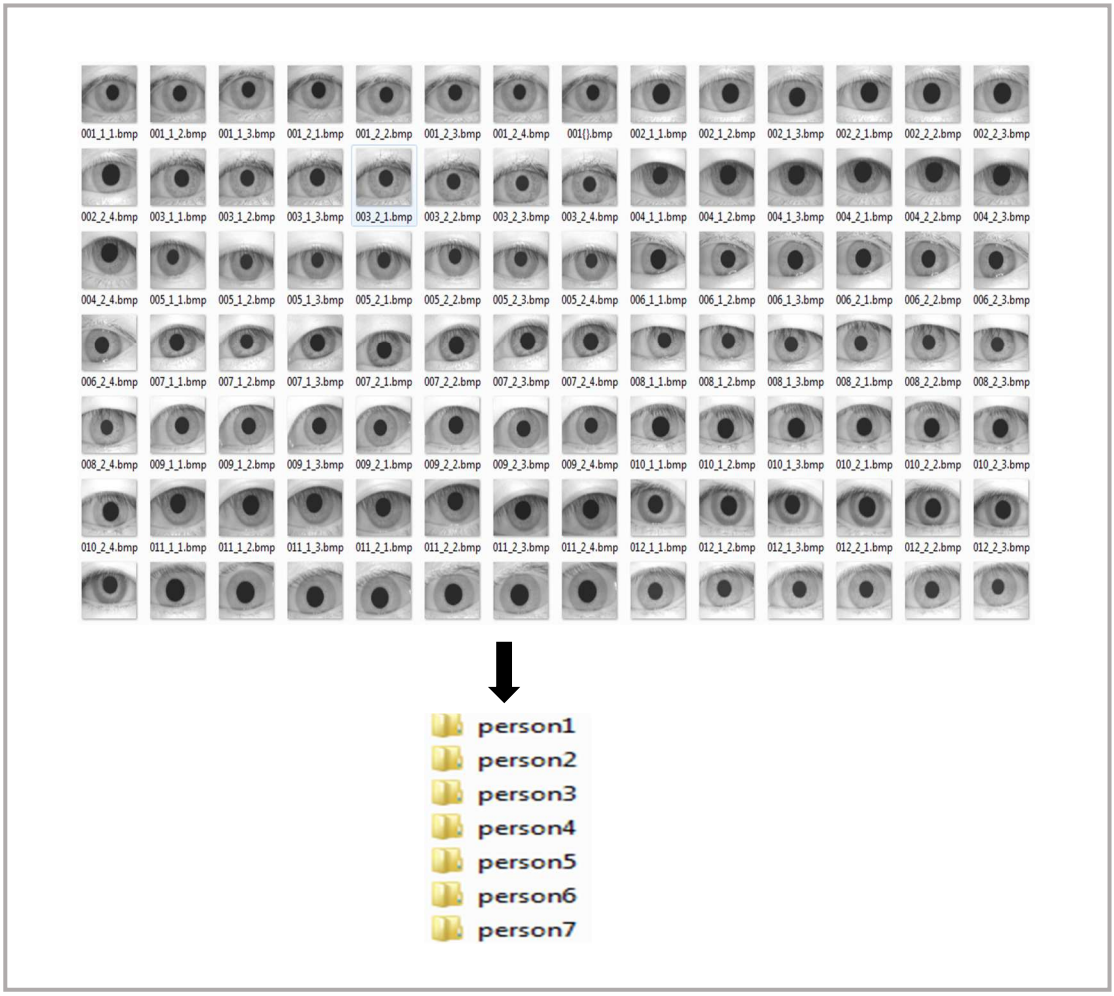


Figure 3 la base de données Casia v1

3.3 Les prétraitements des images :

3.3.1 Amélioration du contraste:

On a utilisé la méthode de l'égalisation d'histogramme vue en cours est un cas particulier où l'histogramme cible possède une distribution uniforme d'intensité. Cette méthode consiste à appliquer une transformation T indépendamment sur chaque pixel de l'image. Cette transformation est construite à partir de l'histogramme cumulé [2].

Pour une image $\{x\}$, on définit n_k le nombre d'occurrences du niveau x_k. La probabilité d'occurrence d'un pixel de niveau x_k dans l'image est :

$$p_x(x_k) = p(x = x_k) = \frac{n_k}{n},$$

Avec n le nombre total de pixels de l'image, et p_x l'histogramme normalisé sur [0, 1].

La transformation T qui a chaque pixel de valeur x_k de l'image d'origine associe une nouvelle valeur s_k, $s_k = T(x_k)$ est alors définie par

$$T(x_k) = (L-1)\sum_{j=0}^{k} p_x(x_j)$$

Où

$$\sum_{j=0}^{k} p_x(x_j)$$

Qui est l'histogramme cumulé.

- Voici ce code qu'on a utilisé obtenir l'histogramme des images ainsi que la formule pour calculer la somme cumulée :

```
# create our own histogram function
def get_histogram(image, bins):
    # array with size of bins, set to zeros
    histogram = np.zeros(bins)

    # loop through pixels and sum up counts of pixels
    for pixel in image:
        histogram[pixel] += 1

    # return our final result
    return histogram

hist = get_histogram(flat, 256)
plt.plot(hist)
# create our cumulative sum function
def cumsum(a):
    a = iter(a)
    b = [next(a)]
    for i in a:
        b.append(b[-1] + i)
    return np.array(b)

# execute the fn
cs = cumsum(hist)
```

Figure 4 code source pour obtenir l'histogramme des images

➢ Ce code montre l'application de la méthode d'égalisation d'histogramme :

```
# numerator & denomenator
nj = (cs - cs.min()) * 255
N = cs.max() - cs.min()
n=abs(cs*255)

# re-normalize the cdf
cs = nj / N

# cast it back to uint8 since we can't use floating point values in images
cs = cs.astype('uint8')


# get the value from cumulative sum for every index in flat, and set that as img_new
img_new = cs[flat]

# put array back into original shape since we flattened it
img_new = np.reshape(img_new, img.shape)

flat2 = img_new.flatten()
histnew = get_histogram(flat2, 256)
plt.plot(histnew)
```

Figure 5 code de la méthode d'égalisation d'histogramme

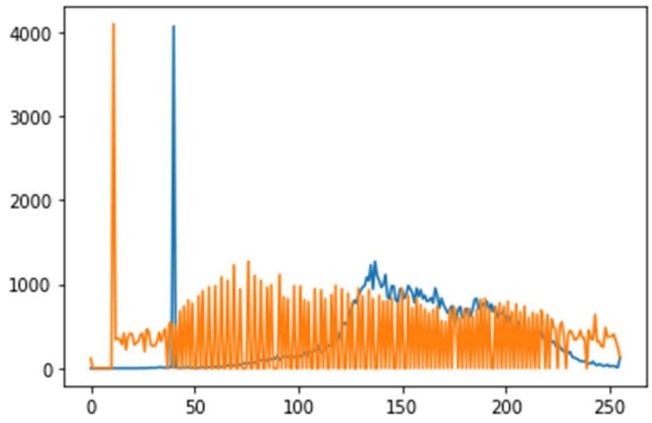


Figure 6 l'histogramme

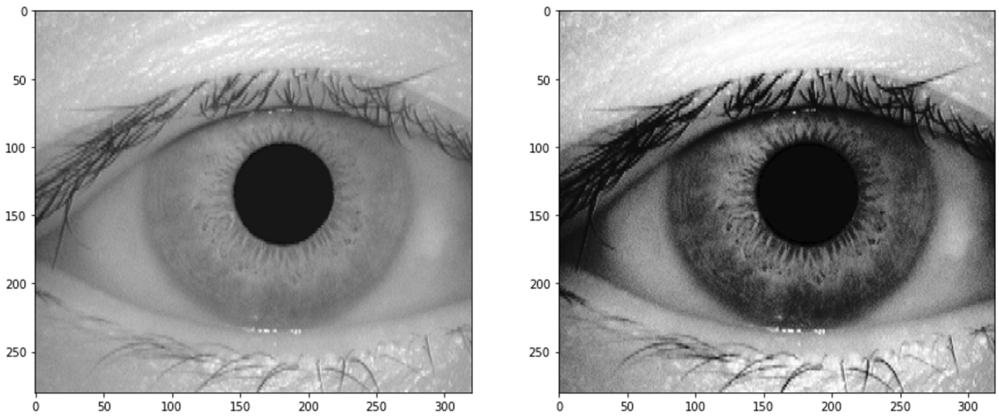


Figure 7 L'image avant et après l'amélioration du contraste

3.3.2 Lissage des images:

L'application du filtre Gaussien:

En une dimension le filtre de Gauss a une réponse impulsionnelle qui est de la forme suivante :

$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

où x représente la distance de l'origine sur l'axe des abscisses, y la distance de l'origine sur l'axe des ordonnées et σ la déviation standard de la distribution gaussienne.

```
#fonction pour appliquer le filtre gaussien H sur un vecteur X
#X=np.array(img)
def convolution2D(X,H,moitie):
    #calculer les dimension de vecteur de l'image
    s = X.shape
    py = int((H.shape[0]-1)/2)
    px = int((H.shape[1]-1)/2)
    Y = X.copy()
    if moitie:
        imax = int(s[1]/2)
    else:
        imax = int(s[1]-px)
    #boucle pour sommer le produit de vecteur par le filtre
    for i in range(px,imax):
        for j in range(py,s[0]-py):
            somme = 0.0
            for k in range(-px,px+1):
                for l in range(-py,py+1):
                    somme += X[j+l][i+k]*H[l+py][k+px]
            Y[j][i] = somme
    return Y
#fonction pour calculer le filtre de dimention P
def filtreGaussien(P):
    epsilon = 0.05
    #calculer sigma
    sigma = P*1.0/math.sqrt(-2*math.log(epsilon))
    h = np.zeros((2*P+1,2*P+1))
    som = 0
    for m in range(-P,P+1):
        for n in range(-P,P+1):
            h[m+P][n+P] = math.exp(-(n*n+m*m)/(2*sigma*sigma))
            som += h[m+P][n+P]
    h = h/som
    return h
```

Figure 8 code source de filtre gaussien

L'image resultant de fitre gaussien(lissage de l'images):

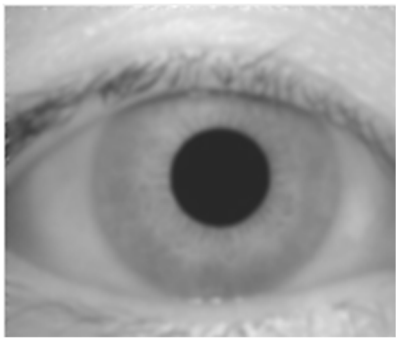


Figure 9 résultat de l'application de filtre gaussien

3.3.3 Segmentation par clustering avec Kmeans :

La méthode des *k*-means a été très utilisée, d'une part pour sa simplicité de mise en oeuvre et d'autre part car elle peut fournir une bonne approximation de la segmentation recherchée.

- On a utilisé l'algorithme kmeans pour effectuer la segmentation des images et voici une petite capture sur le code utilisé.

```
#fonction converger la matrice de k points a une liste de k points
def converged(centroids, old_centroids):
    if len(old_centroids) == 0:
        return False
    if len(centroids) <= 5:
        a = 1
    elif len(centroids) <= 10:
        a = 2
    else:
        a = 4
    for i in range(0, len(centroids)):
        cent = centroids[i]
        old_cent = old_centroids[i]

        if ((int(old_cent[0]) - a) <= cent[0] <= (int(old_cent[0]) + a)) and
            ((int(old_cent[1]) - a) <= cent[1] <= (int(old_cent[1]) + a)) and
            ((int(old_cent[2]) - a) <= cent[2] <= (int(old_cent[2]) + a)):
            continue
        else:
            return False
    return True
#fonction pour recuperer le minimum entre pixel et centroids
def getMin(pixel, centroids):
    minDist = 9999
    minIndex = 0
    for i in range(0, len(centroids)):
        d = numpy.sqrt(int((centroids[i][0] - pixel[0]))**2 + int((centroids[i][1] -
        pixel[1]))**2 + int((centroids[i][2] - pixel[2]))**2)
        if d < minDist:
            minDist = d
            minIndex = i
    return minIndex
```

Figure 10 code source de segmentation1

```
# fonction pour ajuster le centroids
def adjustCentroids(centroids, clusters):
    new_centroids = []
    keys = sorted(clusters.keys())
    for k in keys:
        n = numpy.mean(clusters[k], axis=0)
        new = (int(n[0]), int(n[1]), int(n[2]))
        print(str(k) + ": " + str(new))
        new_centroids.append(new)
    return new_centroids
# fonction pour appliquer le Kmeans
def startKmeans(someK):
    centroids =[(27, 27, 27), (170, 170, 170), (142, 142, 142), (0, 0, 0), (73, 73, 73),
            (123, 123, 123), (0, 0, 0)]
    old_centroids = []
    rgb_range = ImageStat.Stat(im).extrema
    i = 1
    while not converged(centroids, old_centroids) and i <= 20:
        print("Iteration #" + str(i))
        i += 1
        old_centroids = centroids
        clusters = assignPixels(centroids)
        centroids = adjustCentroids(old_centroids, clusters)
    return centroids
# fonction qui utilise l'image originale et qui appelle les autres fonction
def drawWindow(result):
    img = Image.new('RGB', (img_width, img_height), "white")
    p = img.load()
    for x in range(img.size[0]):
        for y in range(img.size[1]):
            RGB_value = result[getMin(px[x, y], result)]
            p[x, y] = RGB_value
    return img
```

Figure 11 code source de segmentation2

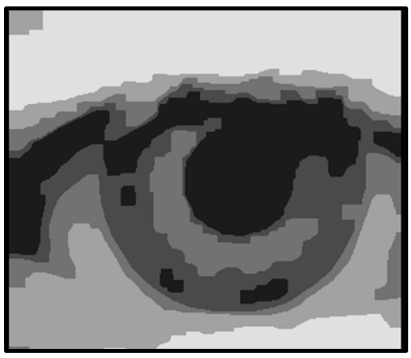


Figure 12 segmentation de l'image

3.3.4 Operations morphologiques :

Application d'opération l'érosion:

L'érosion réduit les ensembles de pixels blancs connectés (pixels à 1) dans une image binaire. Cette opération peut être utilisée pour réduire les forms, enlever des ponts des branches et des petites protuberances

$$Y = X \ominus F$$

```python
#fonction pour effectuer l'opération de l'erosion sur une image
def erosion(image, template_side_length, template):
    #crée une nouvelle image avec les meme dimension de l'ancien image
    new_image = np.zeros(image.shape, image.dtype)
    #template_side_length = le size de filtre
    template_space = calculate_template_space(template_side_length)
    #calculer la moitier de size de filtre
    half_template = int((template_side_length - 1) / 2)
    #boucle pour appliquer le principe d'erosion sur chaque pixel
    #parcourir les lignes et les colonnes de l'image
    for x in range(template_space, new_image.shape[1] - template_space):
        for y in range(template_space, new_image.shape[0] - template_space)
            minimum = 256
             #parcourir les lignes et les colonnes de filtre
            for c in range(0, template_side_length):
                for d in range(0, template_side_length):
                    a = x - half_template - 1 + c
                    b = y - half_template - 1 + d
                    sub = image[b, a] - template[d, c]
                    if sub < minimum:
                        if sub > 0:
                            minimum = sub
            new_image[y, x] = int(minimum)
    return new_image
```

Figure 13 code source de l'érosion

L'image résultat de l'opération de l'érosion :

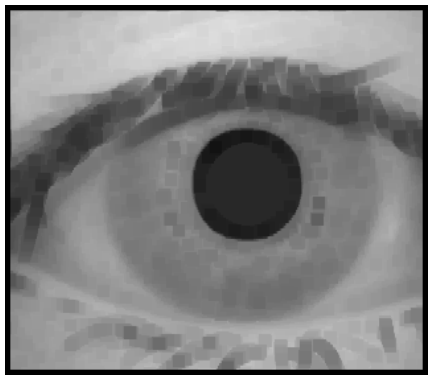


Figure 14 resultat de l'opération de l'érosion

Application d'opération de la dilatation :

La dilatation accroît les ensembles de pixels blancs connectés (pixels à 1) dans une image binaire. Cette opération peut être utilisée pour Accroitre des formes, Combler les trous et les brèches.

$$Y = X \oplus F_2$$

```
#fonction pour effectuer l'opération de dilatation sur une image
def dilation(image, template_side_length, template):
    #crée une nouvelle image avec les meme dimension de l'ancien image
    new_image = np.zeros(image.shape, image.dtype)
    #template_side_length = le size de filtre
    template_space = calculate_template_space(template_side_length)
    #calculer la moitier de size de filtre
    half_template = int((template_side_length - 1) / 2)
    #boucle pour appliquer le principe de dilatation sur chaque pixel
    #parcourir les lignes et les colonnes de l'image
    for x in range(template_space, new_image.shape[1] - template_space):
        for y in range(template_space, new_image.shape[0] - template_space)
            maximum = 0
            #parcourir les lignes et les colonnes de filtre
            for c in range(0, template_side_length):
                for d in range(0, template_side_length):
                    a = x - half_template - 1 + c
                    b = y - half_template - 1 + d
                    sub = image[b, a] - template[d, c]
                    if sub > maximum:
                        if sub > 0:
                            maximum = sub
            new_image[y, x] = int(maximum)
    return new_image
```

Figure 15 code source de dilatation

L'image résultat de l'opération de dilatation:

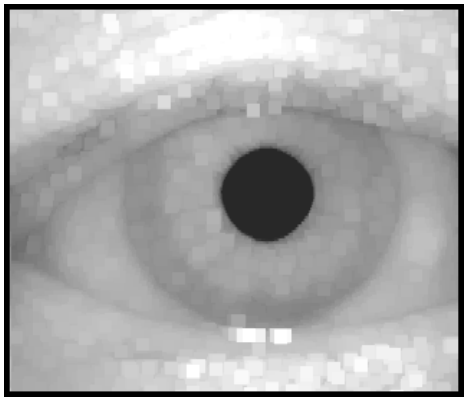


Figure 16 résultat de dilatation

Application d'opération de l'opération d'ouverture :

L'ouverture est définie comme une érosion suivie d'une dilatation utilisant le même élément structurant. L'effet de base d'une ouverture est similaire à l'érosion.

$$X \circ F$$

```
#fonction pour effectuer l'opération d'ouverture(open) sur une image
def open_op(image, template_side_length, template):
    #l'ouverture = dilatation(erosion(image,filtre),filtre)
    #template_side_length=taille de filtre
    new_image = erosion(image, template_side_length, template)
    new_image_2 = dilation(new_image, template_side_length, template)
    return new_image_2
```

Figure 17 code source de l'ouverture

L'image résultat de l'opération d'ouverture :

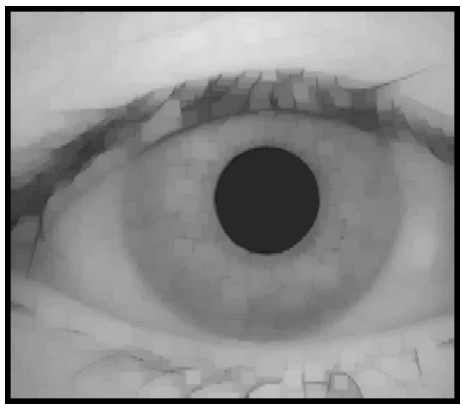


Figure 18 résultat de l'opération d'ouverture

Application d'opération de l'opération de fermeture :

La fermeture est définie comme une dilatation suivie d'une érosion utilisant le même élément structurant. La fermeture est une ouverture effectuée en sens inverse

$$X \bullet F$$

```
#fonction pour effectuer l'opération de fermeture(close) sur une image
def close_op(image, template_side_length, template):
    #fermeture = erosion(dilatation(image,filtre),filtre)
    #template_side_length=taille de filtre
    new_image = dilation(image, template_side_length, template)
    new_image_2 = erosion(new_image, template_side_length, template)
    return new_image_2
```

Figure 19 code source de l'opération de fermeture

L'image résultat de l'opération de l'opération de fermeture :

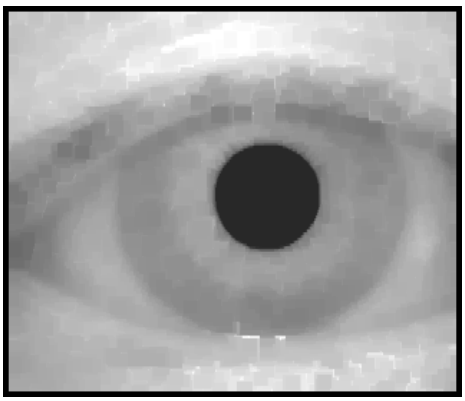


Figure 20 résultat de l'opération de fermeture

L'application de lissage morphologique :

C'est l'application de la fermeture sur le résultat de l'ouverture.

```
#fonction pour effectuer l'opération de lissage morphologique sur l'image
def lissage(image, template_side_length, template):
    #lissage morphologique=fermeture(ouverture(image,filtre),filtre)
    #template_side_length=taille de filtre
    new_image1=open_op(image, template_side_length, template)
    new_image2=close_op(new_image1, template_side_length, template)
    return new_image2
```

Figure 21 code source de l'opération de lissage morphologiques

L'image résultat de l'opération de lissage morphologique :

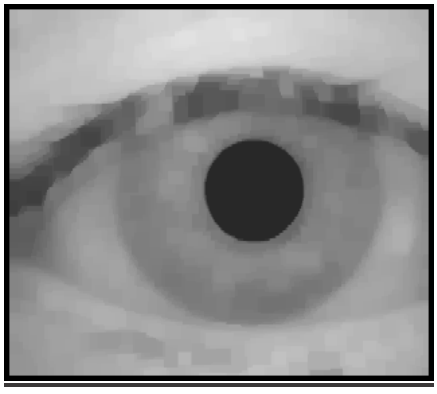


Figure 22 résultat de l'opération morphologiques

3.3.5 Détecteur SIFT et Descripteur SIFT :

L'algorithme SIFT se divise en plusieurs étapes, que nous avons appliquées dans le code :

- **Détection** : création de l'espace des échelles, localisation des points d'intérêt
- **Description** : assignation d'orientation, création des descripteurs

Le but de cette étape est d'obtenir une représentation permettant d'identifier une personne de manière unique et servira par la suite dans l'étape de la mise en correspondance (Matching). [3]

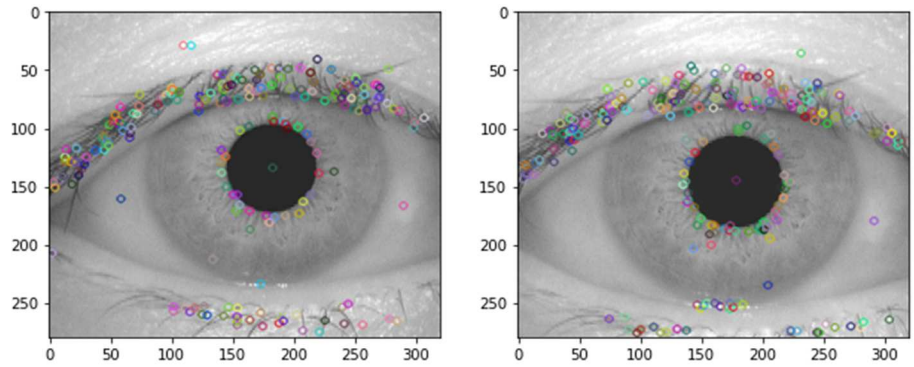


Figure 23 résultat de détecteur sift

➢ Ce code montre l'utilisation de la bibliothèque opencv pour appliquer L'algorithme SIFT sur nos images ;

```
import cv2 as cv
import matplotlib.pyplot as plt
import mysql.connector
import keyword
import re
from PIL import Image
import numpy as np
f=[]
f2=[]
#connection a la base de données "iris_base"
conn = mysql.connector.connect(host="localhost",user="root",password="",
                               database="iris_base")
cursor = conn.cursor()
#créer sift
sift = cv.SIFT_create()
#ouvrirfichier data2 qui contient les noms de toutes les images
fil=open('data2.txt', 'r')
regex = re.compile(r'[\n\r\t]')
#boucle pour parcourir toutes les lignes de fil et les mettre dans une liste
for j in fil:
    f.append(j)
for s in f:
    s = regex.sub("", s)
    f2.append(s)
#parocurir la liste f2 afin de récuperer
for g in f2:
    a='C:/Users/pc/.spyder-py3/tin/imageApresTraitement/'+g
    img1 = cv.imread(a,cv.IMREAD_GRAYSCALE)
    #utiliser le detecteur sift pour récuperer keypoints et descriptors
    kp1, des1 = sift.detectAndCompute(img1,None)
    d=des1.tolist()
    d1=str(d)
    k=cv.KeyPoint_convert(kp1)
    k1=k.tolist()
    k2=str(k1)
    #insérer les resultat de détecteur sift dans la table "iris" de la BDD
    cursor.execute("INSERT INTO iris(image, keypoints , descriptors) VALUES
                   (%s, %s, %s)",(g,k2 ,d1 ))
    conn.commit()
```

Figure 24 code source de l'algorithme sift

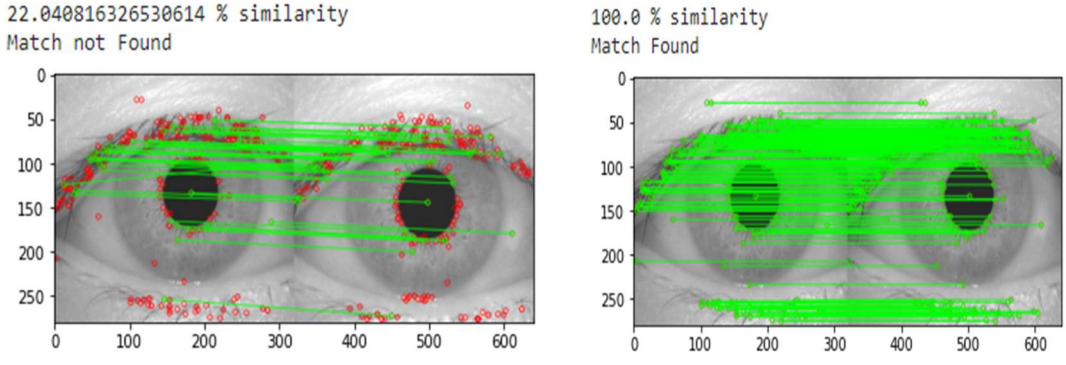
Les résultat de l'opération de détecteur sift sera stocker dans une base de données appelé « iris_base » qui contient la table « iris » qui se compose de quatre colonne comme il est illustré dans la figure 23 ci-dessous:

	id	image	keypoints	descriptors
Éditer Copier Supprimer	1	[BLOB - 11 o]	[[6.10853385925293, 60.34978103637695], [6.1363444...	[[31.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 203.0, ...
Éditer Copier Supprimer	2	[BLOB - 11 o]	[[6.128269195556641, 223.3418731689453], [6.186810...	[[36.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 191.0, ...
Éditer Copier Supprimer	3	[BLOB - 11 o]	[[6.125700950622559, 198.34701538085938], [6.16083...	[[33.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 192.0, ...
Éditer Copier Supprimer	4	[BLOB - 11 o]	[[5.959710121154785, 195.06857299804688], [6.10654...	[[56.0, 13.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 173.0,...
Éditer Copier Supprimer	5	[BLOB - 11 o]	[[6.21353816986084, 227.15821838378906], [6.216782...	[[22.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 187.0, ...
Éditer Copier Supprimer	6	[BLOB - 11 o]	[[6.137049674987793, 227.40887451171875], [9.79430...	[[29.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 205.0, ...
Éditer Copier Supprimer	7	[BLOB - 11 o]	[[5.853448867797852, 68.56205749511719], [6.103432...	[[34.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 201.0, ...
Éditer Copier Supprimer	8	[BLOB - 11 o]	[[5.638632297515869, 216.17276000976562], [6.02952...	[[32.0, 5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 163.0, ...
Éditer Copier Supprimer	9	[BLOB - 11 o]	[[6.113712310791016, 43.34748840332031], [6.156102...	[[32.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 201.0, ...
Éditer Copier Supprimer	10	[BLOB - 11 o]	[[6.114348411560059, 46.347129821777344], [6.13683...	[[32.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 201.0, ...

Figure 25 le résultat de détecteur sift

3.3.6 Mise en correspondance:

Après avoir enregistré avec succès les caractéristiques uniques de chaque image et les avoir stockées dans une structure de données appropriée, il est temps pour nous de récupérer des informations en recherchant un certain fichier image ou un index stocké dans la structure de données et correspondre avec l'image actuelle en examen. C'est-à-dire que différents modèles contenant différentes informations sont stockés dans la bdd, puis nous faisons correspondre l'image sous observation avec tous les modèles stockés et vérifions le pourcentage correspondant et nous annonçons l'image avec le pourcentage le plus élevé de rapport correspondant, comme l'image probable complémentaire de l'image observée actuellement.



Deux images déférentes Deux images similaires

Figure 26 résultat de Mise en correspondance entre deux images

➢ Ce code montre comment nous avons appliqué la mise en correspondance entre deux images (Matching) et le pourcentage de similarité

```
def convert_pts_to_keypoints(pts, size=1):
    kps = []
    if pts is not None:
        if pts.ndim > 2:
            # convert matrix [Nx1x2] of pts into list of keypoints
            kps = [ cv.KeyPoint(p[0][0], p[0][1], _size=size) for p in pts ]
        else:
            # convert matrix [Nx2] of pts into list of keypoints
            kps = [ cv.KeyPoint(p[0], p[1], _size=size) for p in pts ]
    return kps
def identifier(image):
    conn = mysql.connector.connect(host="localhost",user="root",password="", database="iris_base")
    cursor = conn.cursor()
    img1 = cv.imread(image,cv.IMREAD_GRAYSCALE)
    sift = cv.SIFT_create()
    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1,None)

    cursor.execute("""SELECT image, keypoints , descriptors FROM iris """)
    rows = cursor.fetchall()
```

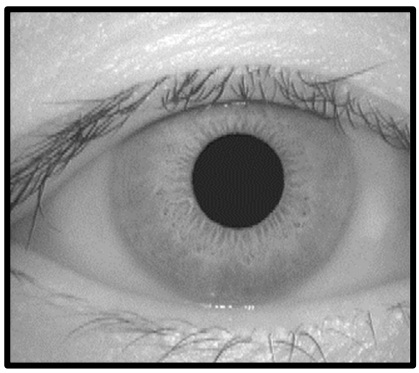
Figure 27 code source de matching1

```
j=0
for row in rows:
    j=j+1
    print(j)
    img2=cv.imread(row[0],cv.IMREAD_GRAYSCALE)
    kp2, des2 = sift.detectAndCompute(img2,None)
    kp2=convert_pts_to_keypoints(np.array(eval(row[1])), size=1)
    des2=np.float32(eval(row[2]))
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks=50)   # or pass empty dictionary

    flann = cv.FlannBasedMatcher(index_params,search_params)

    matches = flann.knnMatch(des1,des2,k=2)

    # Apply ratio test
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append([m])
            a=len(good)
    percent=(a*100)/len(kp2)
    p="{} % similaire".format(percent)
    if percent >= 75.00:
        print('Existe')
        z='Exist'
        break
    else:
        z='N\'existe pas'
        print(z)
return z,p
```

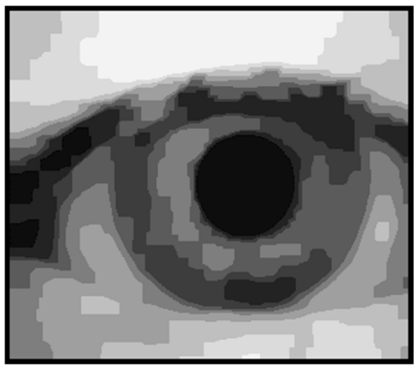
Figure 28 code source de matching2

3.4 Le résultat de prétraitement :

Après l'application de toutes les prétraitements précédents sur les images de la base de données Casia v1 on a obtenu ce résultat qui une image qui montre que les parties les plus importantes de l'image originale 001_1_1.bmp :



L'image originale 001_1_1.bmp



l'image après les prétraitements

Figure 29 résultat de prétraitement appliqué sur une image de la base

4 Les captures d'écran sur l'application :

4.1 L'interface

C'est la première page de système de détection de l'individu par l'iris, elle contient deux boutons :

1- Bouton pour aller à la page de vérification
2- Bouton pour aller à la page de traitement

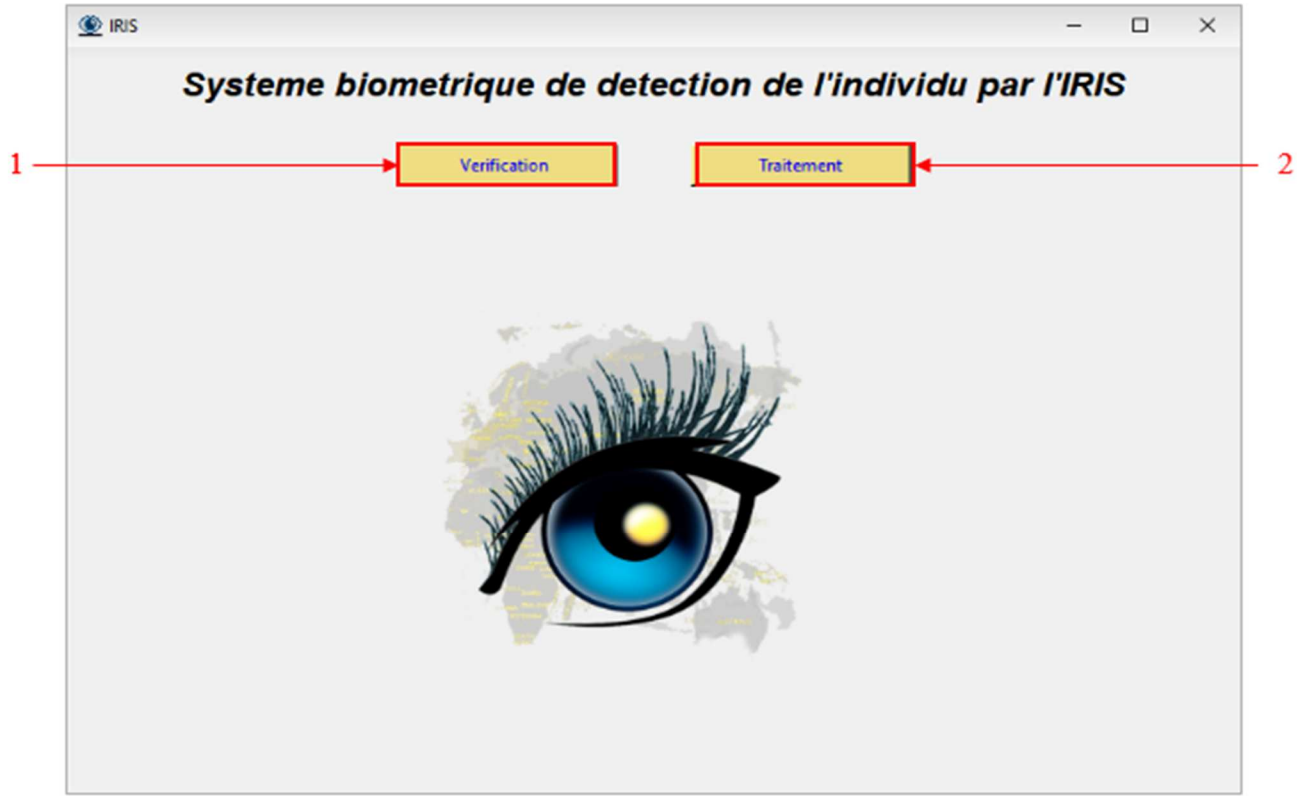


Figure 30 l'interface de système biométriques de détection de l'individu par l'iris

4.2 La page de vérification

Après que l'utilisateur clique sur le bouton 1 de la figure 29 ci-dessus l'utilisateur sera redirigé à la page de vérification illustré par la figure 30 ci-dessous, dans cette page on a quatre boutons :

1- Bouton pour retourner à la page précédente illustré par la figure 29.
2- Bouton pour choisir une image, et on a au-dessous de cette bouton deux cadre vide un pour l'image que l'utilisateur choisi et un pour le résultat après l'application de traitement sur l'image.
3- Bouton pour effectuer les traitements sur l'image choisi par l'utilisateur
4- Bouton pour voir la décision si l'individu existe ou non (Matching, mise en correspondance entre l'image que l'utilisateur a choisie et les images de la base de données).

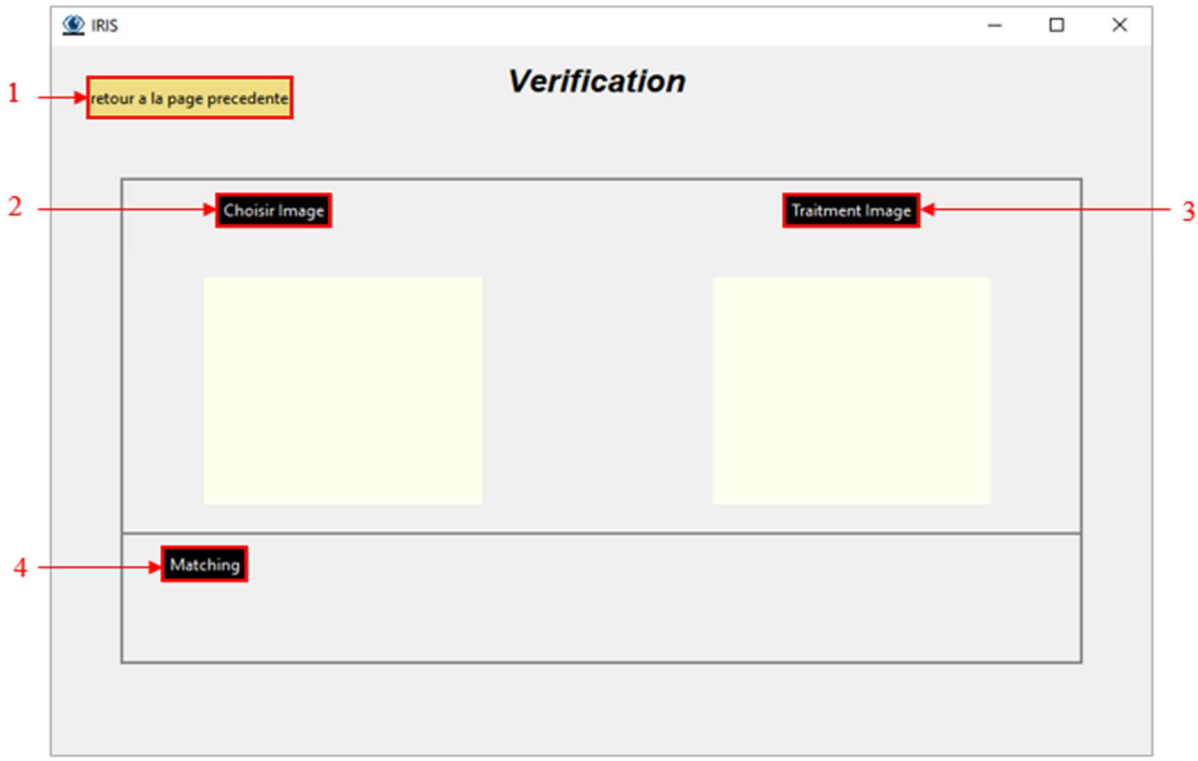


Figure 31 page de vérification de l'individu

Après que l'utilisateur choisi l'image, il clique sur le bouton 3 pour appliquer les traitements sur l'image, et une fenêtre d'informations sera afficher parce que le script python prend un peu du temps pour afficher le résultat.

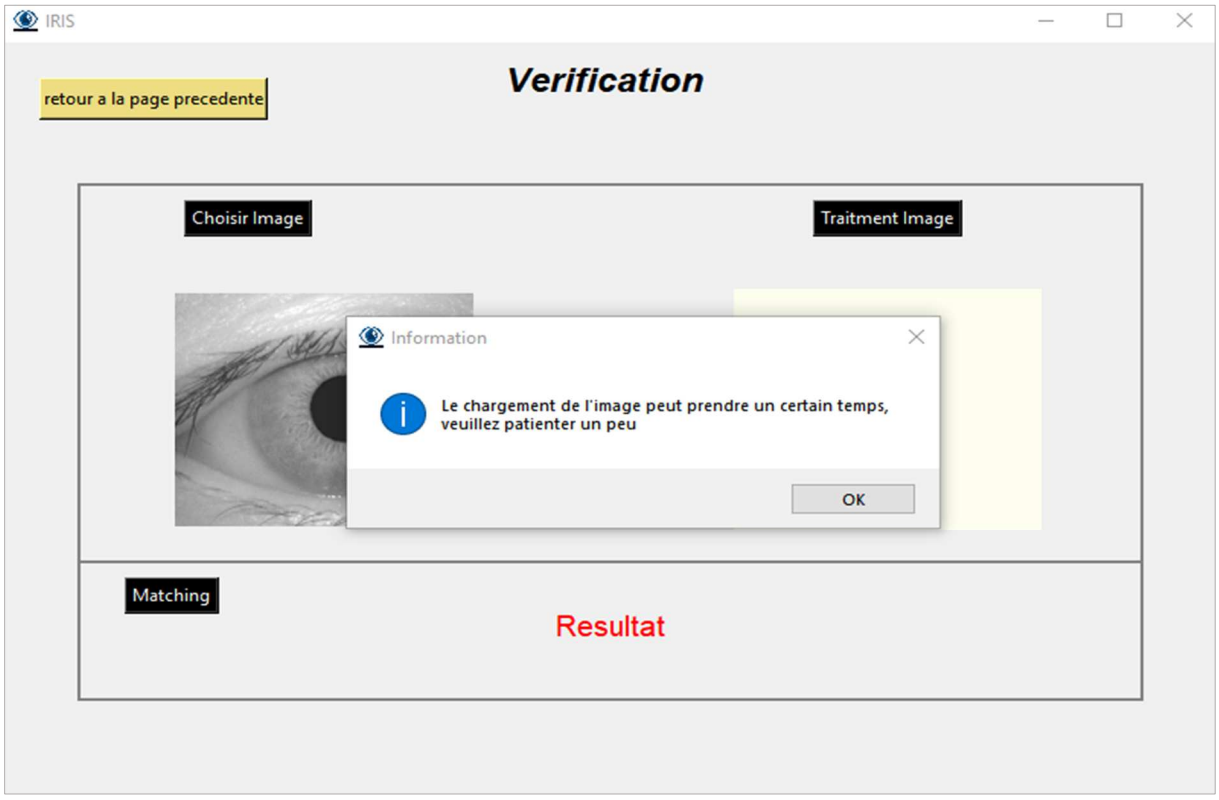


Figure 32 page de vérification de l'individu 2

Après que l'utilisateur clique sur le bouton 3 de la figure 30 une image résultante sera afficher comme il est illustré par 1 dans la figure ci-dessous, et après quand il clique sur le bouton Matching il peut tomber sur deux cas :

1er cas :

Si les caractéristiques de l'image que l'individu a choisie correspond aux caractéristiques d'une autre image stocker dans la base de données un message résultat sera affiché comme il est illustré par 2 dans la figure ci-dessous.

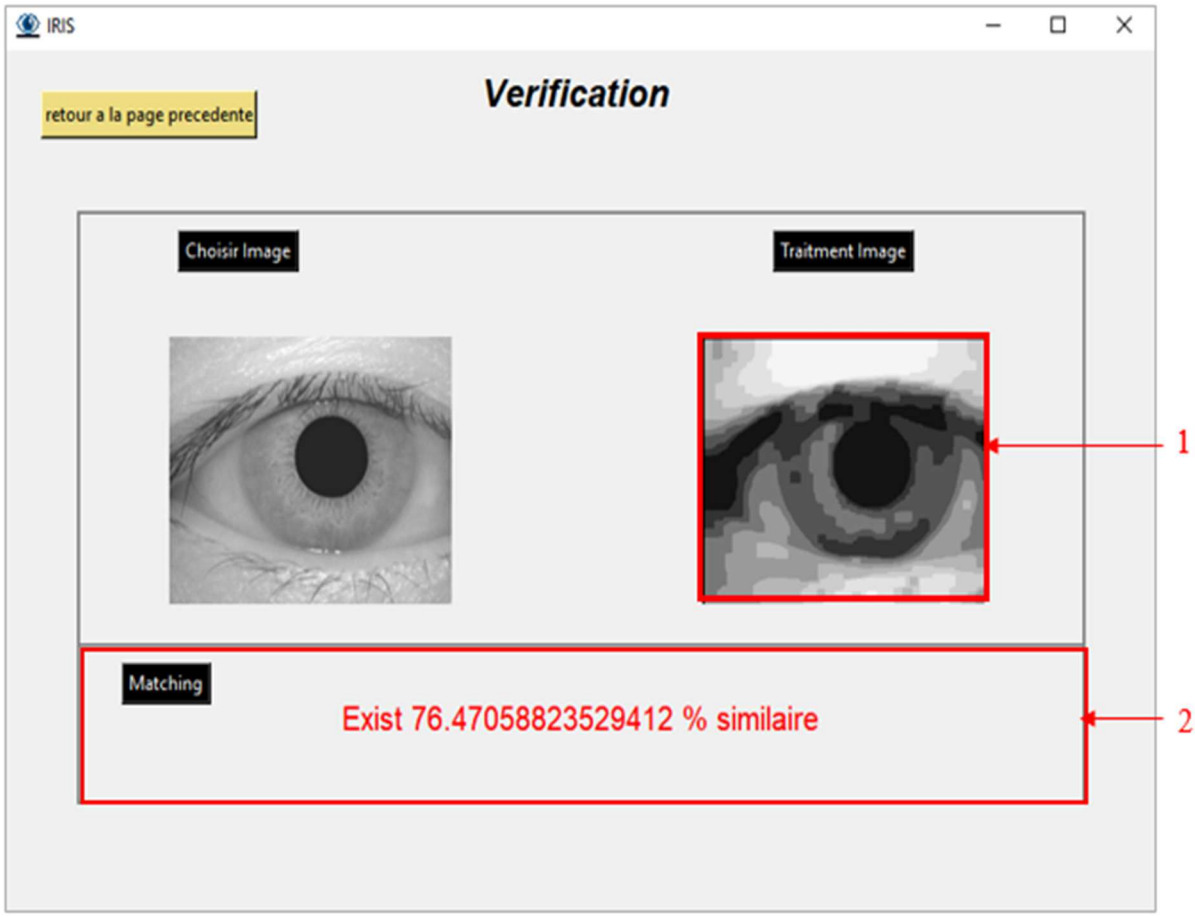


Figure 33 page de vérification de l'individu 3

2éme cas :

Si les caractéristiques de l'image que l'individu a choisie ne correspond pas aux caractéristiques d'une autre image stocker dans la base de données un message résultat sera affiché comme il est illustré par 1 dans la figure ci-dessous.

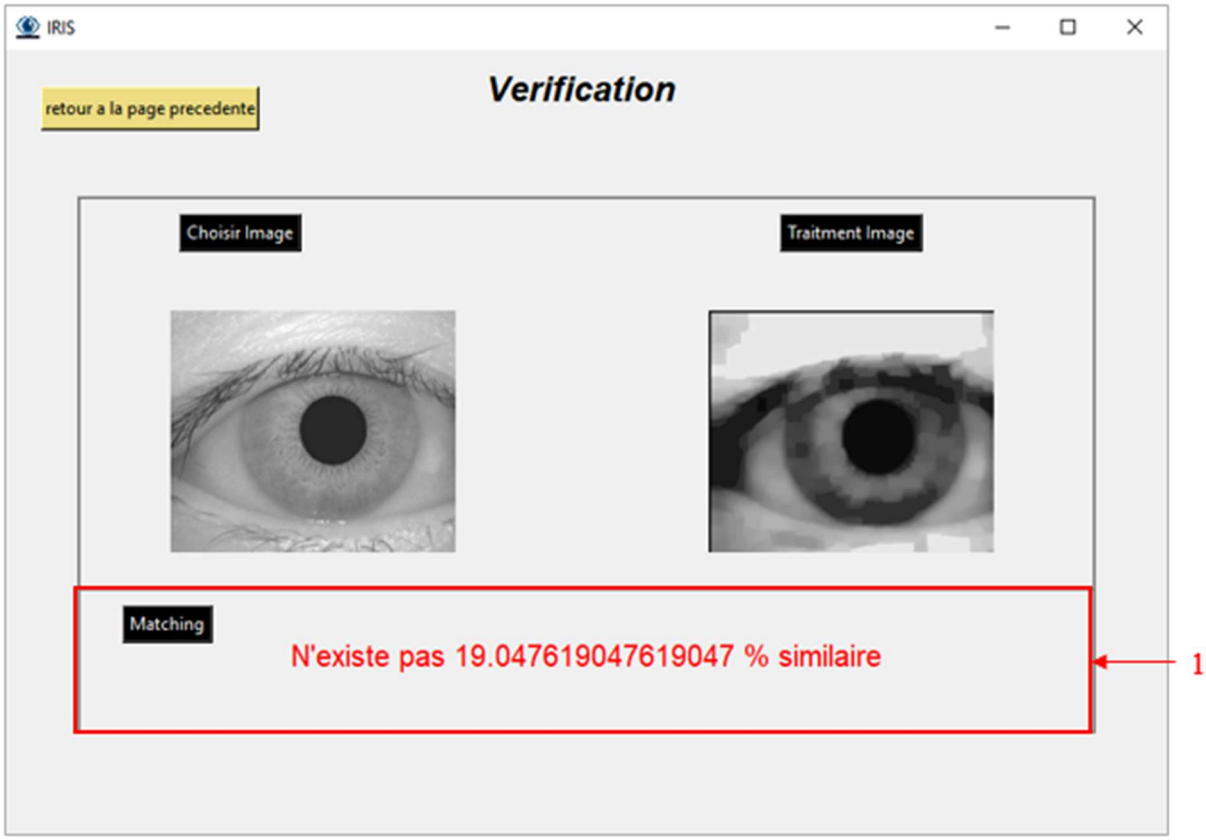


Figure 34 page de vérification de l'individu 4

4.3 La page de traitement

Après que l'utilisateur clique sur le bouton 1 de la figure 30 l'utilisateur sera redirigé à la page de l'interface figure 29 et après qu'il clique sur le bouton 2 de la figure 29 il sera redirigé vers la page de traitement illustré par la figure X ci-dessous, dans cette page on a huit boutons:

1- Bouton pour retourner à la page précédente illustré par la figure 28.
2- Bouton pour choisir une image, et on a au-dessous de cette bouton deux cadre vide un pour l'image que l'utilisateur choisi et un pour le résultat après l'application de traitement sur l'image.
3- Bouton pour appliquer le lissage par filtre gaussien sur l'image
4- Bouton pour appliquer l'ajustement de contraste sur l'image
5- Bouton pour appliquer la segmentation par K = 9
6- Bouton pour appliquer la dilatation sur l'image
7- Bouton pour appliquer l'érosion sur l'image
8- Bouton pour appliquer les lissages morphologiques sur l'image

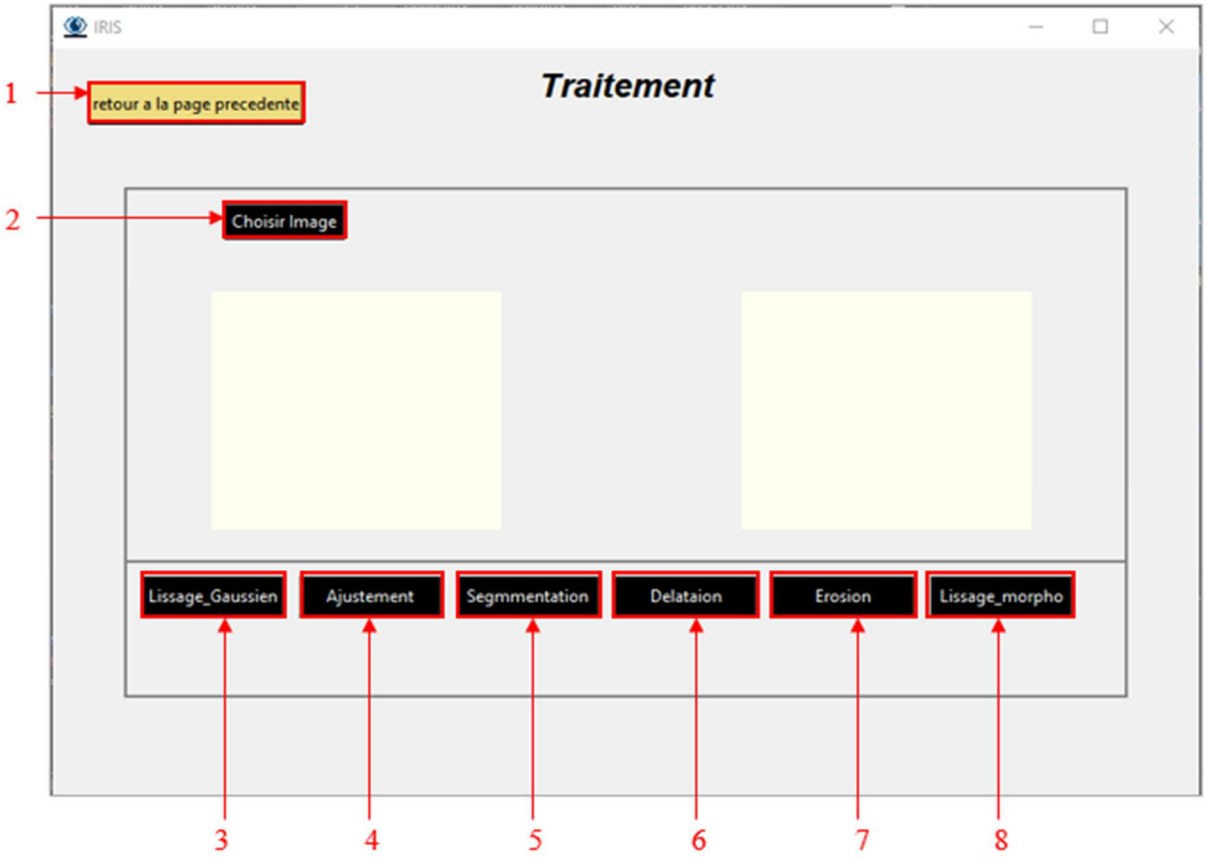


Figure 35 page de traitement

Si l'utilisateur clique sur le bouton 1 de l'image ci-dessous une image résultante sera affichée dans la page comme il est illustré dans la figure ci-dessous par 2 :

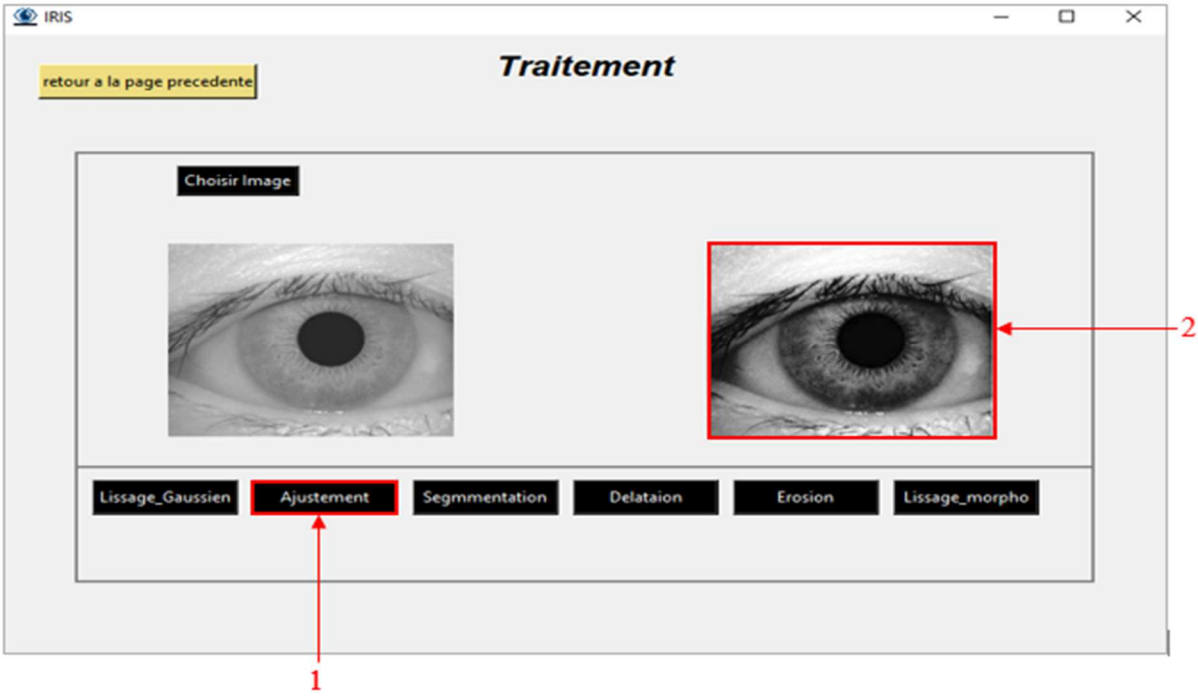
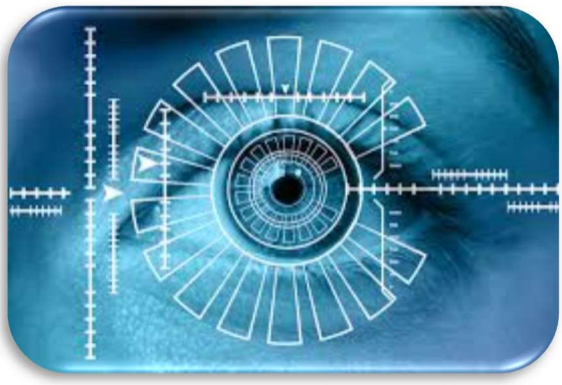


Figure 36 page de traitement 2

5 Conclusion

Dans cette partie du mini-projet nous avons étudié et analyser les images de la base de données casia_v1 afin de pouvoir les utiliser dans notre phase de prétraitement d'images pour ne garder que les parties les plus importantes de l'image et enfin avoir l'identité d'une personne basée sur le degré de similitude entre les caractéristiques extraites et les modèles stockés.

La reconnaissance de l'iris s'est avérée être une mesure de sécurité très utile et polyvalente, largement utilisée à l'avenir.



6 Bibliographie :

[1] https://www.memoireonline.com/03/15/8967/m_Conception-et-mise-en-place-dune-plateforme-de-securisation-par-synthese-et-reconnaissance-biom0.html

[2] http://www.traitement-signal.com/egalisation_d_histogramme.php

[3] https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5053196-decrivez-efficacement-les-features-detectees-avec-sift