Система биометрического считывателя.

Программное обеспечение для ПК.

Текст программы.

РОФ.МГТУ.00001 03

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. *N*дубл. |  |
| Взам. Инв. *N* |  |
| Подп. и дата |  |
| Инв. *N* подп. |  |

Листов 13

Проверил

Рафиков А. Г.

\_\_\_. \_\_\_. 20\_\_\_

Разработал

*­­­*

Пушкин Н. Д.

\_\_\_. \_\_\_. 20\_\_\_

2023

Оглавление

[1 Основная часть 3](#_Toc156900356)

[1.1. Текст программы biometrics.py 3](#_Toc156900357)

[1.2. Текст программы db.py 10](#_Toc156900358)

[1.3. Текст программы secret.py 12](#_Toc156900359)

# 1 Основная часть

## Текст программы biometrics.py

import datetime

import os

import time

import cv2

import face\_recognition

import sys

import numpy

from db import \*

def check\_liveness(eyes\_count, open\_eyes\_count):

"""

Функция сверяет количество найденных открытых глаз и общее число глаз (открытые + закрытые)

Если количества максимально похожи (<5% разницы), то считаем, что проверка на liveness не пройденна

:param eyes\_count: количество найденных открытых глаз

:param open\_eyes\_count: общее число глаз (открытые + закрытые)

"""

return not (abs(eyes\_count - open\_eyes\_count) <= 0.05 \* max(abs(eyes\_count), abs(open\_eyes\_count)))

def add\_registration\_log(log\_index, name):

"""

Метод логгирования регистрации, записи формируются в журнал logs\_registration.txt

:param log\_index: номер события, которое протоколируется

:param name: имя пользователя

"""

messages = [f'[Error] попытка регистрации с существующим именем, имя = {name}\n',

f'[Error] попытка регистрации через фотографию, имя = {name}\n',

f'[Info] успешная регистрация, имя = {name}\n']

"Получаем текущую дату и время"

now = datetime.datetime.now()

"Форматируем дату в нужный формат"

formatted\_date = now.strftime("[%Y-%m-%d %H:%M:%S]")

with open('logs\_registration.txt', 'a') as file:

"Записываем строку в файл"

file.write(f"{formatted\_date} {messages[log\_index]}")

def add\_auth\_log(log\_index, name=''):

"""

Метод логгирования аутентификации, записи формируются в журнал logs\_authentication.txt

:param log\_index: номер события, которое протоколируется

:param name: имя пользователя

"""

messages = [f'[Error] аутентификация не пройдена\n',

f'[Error] попытка аутентификации через фотографию\n',

f'[Info] успешная аутентификация, имя = {name}\n']

" Получаем текущую дату и время"

now = datetime.datetime.now()

"Форматируем дату в нужный формат"

formatted\_date = now.strftime("[%Y-%m-%d %H:%M:%S]")

with open('logs\_authentication.txt', 'a') as file:

"Записываем строку в файл"

file.write(f"{formatted\_date} {messages[log\_index]}\n")

def open\_back\_camera(id):

"""

Метод включения фоновой камеры

:param id: идентификатор камеры

"""

camera = cv2.VideoCapture(id)

while True:

"получение кадров с камеры и настройка размеры окна вывода"

ret, frame = camera.read()

dim = (750, 500)

frame = cv2.resize(frame, dim)

"вывод на окно"

if ret:

cv2.imshow('back camera', frame)

"Точка выхода по кнопке"

if cv2.waitKey(1) == ord('q'):

break

"уничтожение окон"

camera.release()

cv2.destroyAllWindows()

def find\_count\_of\_video():

"""

Получение количества видео, чтобы дать номер следующему

При регистрации записывается видео, из которого

видео после завершения этапа регистрации удаляется, чтобы не хранить персональные данные

"""

video\_path = os.path.join(os.path.dirname(os.path.abspath("\_\_file\_\_")), "videos")

content = os.listdir(video\_path)

indexes = []

for file in content:

"записанное видео имеет номер, вырежем расширение и получим нужный номер"

if os.path.isfile(os.path.join(video\_path, file)) and file.endswith('.mp4'):

"отрежим расширение"

file = file[:-4]

indexes.append(int(file))

return max(indexes) if len(indexes) != 0 else 0

def make\_video(id\_camera, name):

"""

Метод записи видео при регистрации для записи биометрических данных

:param id\_camera: идентификатор камеры, откуда читать изображение

:param name: имя пользователя, введенное при регистрации

:return: номер записанного видео, результат проверки на liveness

"""

"готовые фильтры для опознавания лиц и глаз"

cascade\_path = ('filters/haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier('filters/haarcascade\_eye\_tree\_eyeglasses.xml')

righteye\_cascade = cv2.CascadeClassifier('filters/haarcascade\_righteye\_2splits.xml')

"переменные для подсчета найденных глаз (открытых и закрытых или только открытых)"

all\_eyes\_count = 0

open\_eyes\_count = 0

"на основе этого фильтра создадим классификатор"

classifier = cv2.CascadeClassifier(cascade\_path)

"то откуда читаем видео (камера или видеофайлы), в нашем случае камера"

"индекс камеры 0, так как она одна (встроенная)"

camera = cv2.VideoCapture(id\_camera)

# fps = camera.get(cv2.CAP\_PROP\_FPS) # Получаем кадры в секунду

# print(fps)

# camera = cv2.VideoCapture('rtsp://admin:admin1@192.168.1.2/1')

"переменные времени и размеров окна"

start\_time = datetime.datetime.now()

delta\_time\_seconds = 0

frame\_width = int(camera.get(3))

frame\_height = int(camera.get(4))

"получение количества видео, чтобы дать название новому"

index = find\_count\_of\_video() + 1

size = (frame\_width, frame\_height)

"запись видео"

fourcc = cv2.VideoWriter.fourcc('m', 'p', '4', 'v')

video\_file = cv2.VideoWriter(f'videos/{index}.mp4', fourcc, 10, size)

liveness = False

while delta\_time\_seconds < 7:

"захват видео с декодированием, frame - полученный кадр"

\_, frame = camera.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

"найдем лица на камере, передав туда кадр"

faces = classifier.detectMultiScale(

frame, # кадр

scaleFactor=1.1, # масштабирование

minNeighbors=10, # строгость критерия отбора (5 по документации)

minSize=(30, 30), # минимальный размер

flags=cv2.CASCADE\_SCALE\_IMAGE

)

"Обведем лица прямоугольниками по их размерам и координатам"

"Также цвет рамка в rgb-формате и толщину линии"

for (x, y, w, h) in faces:

roi\_gray = gray[y:y + h, x:x + w]

roi\_color = frame[y:y + h, x:x + w]

cv2.rectangle(frame, (x, y), (x + w, y + h + 20), (0, 255, 0), 2)

cv2.putText(frame, name, (x + 10, y + 15), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 4)

"найдем глаза"

eyes = eye\_cascade.detectMultiScale(roi\_gray, minNeighbors=10)

allEyes = righteye\_cascade.detectMultiScale(roi\_gray, minNeighbors=10)

"выделение открытых глаз"

for (ex, ey, ew, eh) in eyes:

open\_eyes\_count += 1

# cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (255, 0, 0), 2)

"выделение открытых и закрытых глаз"

for (ex, ey, ew, eh) in allEyes:

all\_eyes\_count += 1

# cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 0, 40), 2)

"Для демонстрации результата на кадре"

cv2.imshow('registration', frame)

"проверка на liveness"

liveness = check\_liveness(all\_eyes\_count, open\_eyes\_count)

"сохранение в файл"

if liveness:

video\_file.write(frame)

"Точка выхода по кнопке"

if cv2.waitKey(1) == ord('q'):

break

"изменение времени"

current\_time = datetime.datetime.now()

delta\_time\_seconds = (current\_time - start\_time).total\_seconds()

"прекращение захвата и закрытие окон"

camera.release()

video\_file.release()

cv2.destroyAllWindows()

return index, liveness

def make\_photos\_from\_video(name, video\_number):

"""

из записанного видео делает фотографии из которых будут взяты лица для БД

name = имя пользователя

video\_number - номер видео

"""

cap = cv2.VideoCapture(f"videos/{video\_number}.mp4")

count = 0

"создание папки базы данных"

if not os.path.exists("dataset"):

os.mkdir("dataset")

while True:

"чтение видео"

ret, frame = cap.read()

fps = cap.get(cv2.CAP\_PROP\_FPS)

multiplier = fps \* 2

if ret:

"получение фото из видео"

frame\_id = int(round(cap.get(1)))

k = cv2.waitKey(1)

if frame\_id % multiplier == 0:

"сохранение фото регистрации"

cv2.imwrite(f"dataset/{name}\_{count}.jpg", frame)

count += 1

"выключение демонстрации окна по кнопке"

if k == ord("q"):

break

else:

break

"уничтожение окон"

cap.release()

cv2.destroyAllWindows()

def do\_authentification(id\_camera):

"""

Метод проведения аутентификации

:param id\_camera: идентификатор камеры

:return: результат аудентификации с именем

"""

images = os.listdir("dataset")

while True:

"готовый фильтр для опознавания лиц"

cascade\_path = ('filters/haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier('filters/haarcascade\_eye\_tree\_eyeglasses.xml')

righteye\_cascade = cv2.CascadeClassifier('filters/haarcascade\_righteye\_2splits.xml')

"переменные для подсчета найденных глаз (открытых и закрытых или только открытых)"

all\_eyes\_count = 0

open\_eyes\_count = 0

"на основе этого фильтра создадим классификатор"

classifier = cv2.CascadeClassifier(cascade\_path)

"то откуда читаем видео (камера или видеофайлы), в нашем случае камера"

camera = cv2.VideoCapture(id\_camera)

"захват видео с декодированием, frame - полученный кадр"

camera\_is\_ready, frame = camera.read()

try:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

except:

pass

if camera\_is\_ready:

"нахожеждение лица и его перекодировка в числовой вектор"

faces\_location = face\_recognition.face\_locations(frame)

faces\_encodings = face\_recognition.face\_encodings(frame, faces\_location)

"найдем лица на камере, передав туда кадр"

faces = classifier.detectMultiScale(

frame, # кадр

scaleFactor=1.1, # масштабирование

minNeighbors=10, # строгость критерия отбора (5 по документации)

minSize=(30, 30), # минимальный размер

flags=cv2.CASCADE\_SCALE\_IMAGE

)

for (x, y, w, h) in faces:

roi\_gray = gray[y:y + h, x:x + w]

roi\_color = frame[y:y + h, x:x + w]

"найдем глаза"

eyes = eye\_cascade.detectMultiScale(roi\_gray, minNeighbors=10)

allEyes = righteye\_cascade.detectMultiScale(roi\_gray, minNeighbors=10)

for (ex, ey, ew, eh) in eyes:

open\_eyes\_count += 1

"выделение открытых и закрытых глаз"

for (ex, ey, ew, eh) in allEyes:

all\_eyes\_count += 1

"Для демонстрации результата на кадре"

# cv2.imshow('auth', frame)

"Далее произведем сравнение с датасетом"

result,name = compare\_face\_with\_db(faces\_encodings[0])

if result[0]:

liveness = check\_liveness(open\_eyes\_count, all\_eyes\_count)

return\_values = (True, name, liveness) if liveness else (True, 'photo', liveness)

return return\_values

return False, None, False

def do\_authentification\_with\_name(id):

"""

Метод проведение аутентификации с выводом имени (результата) на экран

:param id: идентификатор камеры

"""

while (True):

"проведение аутентификации"

result, name, liveness = do\_authentification(id)

print(result,name,liveness)

unknown\_name = 'unknown'

"получение имени из результата"

if name is None:

name = unknown\_name

start\_time = datetime.datetime.now()

delta\_time\_seconds = 0

"готовый фильтр для опознавания лиц"

cascade\_path = ('filters/haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier('filters/haarcascade\_eye\_tree\_eyeglasses.xml')

righteye\_cascade = cv2.CascadeClassifier('filters/haarcascade\_righteye\_2splits.xml')

"на основе этого фильтра создадим классификатор"

classifier = cv2.CascadeClassifier(cascade\_path)

"то откуда читаем видео (камера или видеофайлы), в нашем случае камера"

"индекс камеры 0, так как она одна (встроенная)"

camera = cv2.VideoCapture(id)

while delta\_time\_seconds < 5:

"захват видео с декодированием, frame - полученный кадр"

\_, frame = camera.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

"найдем лица на камере, передав туда кадр"

faces = classifier.detectMultiScale(

frame, # кадр

scaleFactor=1.1, # масштабирование

minNeighbors=10, # строгость критерия отбора (5 по документации)

minSize=(30, 30), # минимальный размер

flags=cv2.CASCADE\_SCALE\_IMAGE

)

"Обведем лица прямоугольниками по их размерам и координатам"

"Также цвет рамка в rgb-формате и толщину линии"

for (x, y, w, h) in faces:

roi\_gray = gray[y:y + h, x:x + w]

roi\_color = frame[y:y + h, x:x + w]

cv2.rectangle(frame, (x, y), (x + w, y + h + 20), (0, 255, 0), 2)

cv2.putText(frame, name, (x + 10, y + 15), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 4)

"найдем глаза"

eyes = eye\_cascade.detectMultiScale(roi\_gray, minNeighbors=10)

all\_eyes = righteye\_cascade.detectMultiScale(roi\_gray, minNeighbors=10)

"выделение открытых глаз"

for (ex, ey, ew, eh) in eyes:

cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (255, 0, 0), 2)

"выделение открытых и закрытых глаз"

for (ex, ey, ew, eh) in all\_eyes:

cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 0, 40), 2)

"Для демонстрации результата на кадре"

cv2.imshow('auth', frame)

"Точка выхода по кнопке"

if cv2.waitKey(1) == ord('q'):

break

current\_time = datetime.datetime.now()

delta\_time\_seconds = (current\_time - start\_time).total\_seconds()

"прекращение захвата и закрытие окон"

camera.release()

cv2.destroyAllWindows()

if not liveness:

print(f"проверка на liveness не пройдена")

add\_auth\_log(1)

"если человек"

if result and liveness:

print(f"аутентификации пройдена успешно,это - {name}!")

add\_auth\_log(2, name)

elif liveness:

print(f"неизвестный человек")

add\_auth\_log(0)

time.sleep(1)

def register\_another\_person(id\_camera):

"""

регистрация нового пользователя системы

Делаем видео, из него после делаем скрины лиц и добавляем в наш dataset

:param id\_camera: идентификатор камеры

"""

"ввод имени регистрации"

name = input('введите имя для регистрации:')

name = name.lower()

"путь, куда запишем видео регистрации"

video\_path = os.path.join(os.path.dirname(os.path.abspath("\_\_file\_\_")), "dataset")

content = os.listdir(video\_path)

name\_is\_exist = False

"проверка на наличие имени, среди зарегистрированных"

for file in content:

index = file.index('\_')

if name == file[:index]:

name\_is\_exist = True

break

if name\_is\_exist:

print('пользователь существует, введите другое имя')

add\_registration\_log(0, name)

else:

"Делаем видео и сохраняем"

index, liveness = make\_video(id\_camera, name)

if not liveness:

"если проверка на liveness не прошла"

print('проверка на liveness не пройдена')

add\_registration\_log(1, name)

else:

"Делаем фото из видео"

make\_photos\_from\_video(name, index)

"Удалим видео и фото, предварительно занесем в БД вектор лиц"

add\_photo\_to\_db(name,index)

print('регистрация проведена успешно')

add\_registration\_log(2, name)

def main():

"""

Точка входа в программу:

1 режим: аутентификация - передаваемое видео сопоставляется с датасетом

2 режим: регистрация - запись видео с последующем занесением в датасет

"""

mode = sys.argv[1].lower()

if mode == 'back1':

open\_back\_camera('rtsp://admin:admin1@192.168.1.2/1')

elif mode == 'back2':

open\_back\_camera('rtsp://admin:admin1@192.168.1.3/1')

elif mode == 'back3':

open\_back\_camera(0)

elif mode == 'registration':

register\_another\_person(0)

elif mode == 'auth':

do\_authentification\_with\_name(0)

elif mode == 'registration\_ip':

id = 'rtsp://admin:admin1@192.168.1.2/1'

register\_another\_person(id)

elif mode == 'auth\_ip':

id = 'rtsp://admin:admin1@192.168.1.2/1'

do\_authentification\_with\_name(id)

if \_\_name\_\_ == '\_\_main\_\_':

main()

## Текст программы db.py

import os

import sqlite3

import time

import cv2

import face\_recognition

import pickle

import pysqlcipher3

import secret

def add\_photo\_to\_db(name, index=0):

"""

Добавление в базу данных векторов лиц для переданного имени

Занесение в базу осуществляется в зашифрованном виде

:param name: имя пользователя

:param index: номер видео,которое надо удалить

"""

"удалим видео"

os.remove(f'videos/{index}')

"подсоединимся к базе данных и возьмем управление (курсор)"

connection = sqlite3.connect('database.db')

cursor = connection.cursor()

"подключим шифрование на нашем ключе с помощью модуля pysqlcipher3"

connection.execute(f"ATTACH DATABASE 'database.db' AS encrypted KEY '{secret.encryption\_key}';")

"Создаем таблицу Users если ее нет"

cursor.execute('''

CREATE TABLE IF NOT EXISTS Users (

username TEXT NOT NULL,

face\_vector BLOB)

''')

connection.commit()

"все изображения из видео перенес в ДБ"

images = os.listdir("dataset")

for image in images:

if name in image:

face\_img = face\_recognition.load\_image\_file(f"dataset/{image}")

try:

"его декодирование"

face\_enc = face\_recognition.face\_encodings(face\_img)[0]

"применим сериализацию"

face\_pickle = pickle.dumps(face\_enc)

"добавим запись о пользователе"

cursor.execute('INSERT INTO Users (username, face\_vector) VALUES (?, ?)', (name, face\_pickle))

connection.commit()

"удалим фото"

os.remove(f'dataset/{image}')

except:

pass

connection.close()

def compare\_face\_with\_db(face\_from\_camera):

"""

Функция сравнения векторов лиц из БД и переданного

:param face: вектор лица из видео при аутентификаци

:return: имя пользователя

"""

"подсоединимся к базе данных и возьмем управление (курсор)"

connection = sqlite3.connect('database.db')

cursor = connection.cursor()

"достанем все записи из БД"

cursor.execute('SELECT \* FROM Users')

users = cursor.fetchall()

"сравним с нашим лицом"

for user in users:

try:

face\_pickle = pickle.loads(user[1])

result = face\_recognition.compare\_faces([face\_pickle], face\_from\_camera)

if result == [True]:

return result, user[0]

except:

pass

return False, 'unknown'

## Текст программы secret.py

encryption\_key = "your\_encryption\_key"

**Лист регистрации изменений к РКД**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Содержание изменения / основание / дата внесения изменения | Количество  страниц РКД | Подпись автора РКД |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |