UNIVERSIDADE PAULISTA

IGOR GALDINO RA: N009GC5

LEONARDO MARANA RA: N0087G5

LUKAS ANDRADE NASCIMENTO RA: G4143D4

PEDRO HENRIQUE DE SOUZA PUTINATTI RA: N0329C5

DESENVOLVIMENTO DE UM SISTEMA DE IDENTIFICAÇÃO E AUTENTICAÇÃO BIOMÉTRICA

SANTOS

2024

IGOR GALDINO RA: N009GC5

LEONARDO MARANA RA: N0087G5

LUKAS ANDRADE NASCIMENTO RA: G4143D4

PEDRO HENRIQUE DE SOUZA PUTINATTI RA: N0329C5

DESENVOLVIMENTO DE UM SISTEMA DE IDENTIFICAÇÃO E AUTENTICAÇÃO BIOMÉTRICA

Trabalho de finalização de semestre para obtenção de nota de APS na matéria de Processamento de Imagem e Visão Computacional – PIVC em Ciência da Computação apresentado à Universidade Paulista – UNIP.

Orientador: Prof. Dr. Jose de Franca Bueno

SANTOS

2024

RESUMO

Este trabalho tem como objetivo o desenvolvimento de um sistema de identificação e autenticação biométrica, utilizando impressões digitais para restringir o acesso a informações sensíveis. O sistema foi projetado para autenticar usuários em diferentes níveis de acesso, com base em impressões digitais coletadas de imagens. A solução foi construída utilizando a linguagem Python e algumas bibliotecas essenciais, como OpenCV, Face Recognition, NumPy, e Flask, para processar imagens e realizar a autenticação biométrica. O processo envolve a normalização da imagem da impressão digital, a extração de características e a comparação com descritores previamente armazenados de usuários autorizados. O sistema também inclui uma interface web desenvolvida com Flask, permitindo que os usuários realizem o login e a autenticação através de um arquivo de imagem. O trabalho aborda o desenvolvimento do sistema, desde a captura e processamento da imagem até a autenticação do usuário, com aplicação prática no controle de acesso a informações críticas sobre propriedades rurais e o uso de agrotóxicos.

Palavras-chave: Identificação biométrica; Autenticação biométrica; Impressões digitais; Processamento de imagem; OpenCV; Flask; Python; Reconhecimento de padrões; Segurança de dados; Controle de acesso.

ABSTRACT

This work aims to develop a biometric identification and authentication system, using digital fabrics to restrict access to sensitive information. The system is designed to authenticate users at different access levels based on collected digital generations of images. The solution was built using the Python language and some essential libraries, such as OpenCV, Face Recognition, NumPy, and Flask, to process images and perform biometric authentication. The process involves normalizing the fingerprint image, removing features, and comparing it to previously stored descriptors from authorized users. The system also includes a web interface developed with Flask, allowing users to login and authenticate via an image file. The work addresses the development of the system, from image capture and processing to user authentication, with practical application in controlling access to critical information about rural properties and the use of pesticides.

Keywords: Biometric identification; Biometric authentication; Fingerprints; Image processing; OpenCV; Bottle; Python; Pattern recognition; Data security; Access control.

SUMÁRIO

1 INTRODUÇÃO

O reconhecimento biométrico, particularmente por meio de impressões digitais, tem se consolidado como uma das formas mais seguras e precisas de autenticação de indivíduos. Ao longo dos anos, a tecnologia de reconhecimento de impressões digitais foi aplicada em uma vasta gama de sistemas, desde controles de acesso em dispositivos móveis até autenticações em sistemas de segurança de dados em empresas e órgãos governamentais.

No contexto deste trabalho, o objetivo é desenvolver um sistema de identificação e autenticação biométrica baseado no reconhecimento de impressões digitais. Esse sistema será capaz de verificar a identidade de um usuário com base em sua impressão digital, comparando-a com uma base de dados de impressões digitais previamente cadastradas, a fim de determinar se o usuário tem permissão para acessar determinadas informações. A aplicação desse sistema será voltada principalmente para o controle de acesso a informações sensíveis sobre propriedades rurais e o uso de agrotóxicos, com a função de garantir que apenas usuários autorizados possam visualizar ou modificar dados críticos.

O desenvolvimento deste sistema envolve a combinação de diferentes técnicas de processamento de imagem, aprendizado de máquina e autenticação biométrica. Para isso, foram utilizadas as bibliotecas Python OpenCV, NumPy, Flask e outras, que oferecem ferramentas poderosas para a captura e processamento de imagens, além da criação de uma interface web que facilita a interação do usuário com o sistema.

2. REVISÃO DE LITERATURA

O reconhecimento biométrico é um campo da segurança da informação que visa utilizar características físicas ou comportamentais dos indivíduos para autenticar sua identidade. Uma das formas mais comuns de autenticação biométrica é o uso de impressões digitais. Com a evolução das tecnologias de captura e processamento de imagens, o reconhecimento de impressões digitais tornou-se uma solução prática e eficaz para controle de acesso em sistemas de segurança.

2.1 HISTÓRIA DO RECONHECIMENTO BIOMÉTRICO

O conceito de biometria remonta a séculos atrás, mas foi somente no final do século XX que os avanços em tecnologia permitiram o desenvolvimento de sistemas de reconhecimento biométrico em larga escala. Inicialmente, o reconhecimento de impressões digitais era feito manualmente, com o auxílio de papéis e tintas. No entanto, com o advento da informática e da digitalização de imagens, esse processo passou a ser automatizado, permitindo uma análise mais precisa e rápida.

2.2 TECNOLOGIAS DE PROCESSAMENTO DE IMAGENS PARA RECONHECIMENTO BIOMÉTRICO

O uso de técnicas de processamento de imagem para análise e extração de características de impressões digitais tem sido fundamental para o sucesso dos sistemas de reconhecimento biométrico. O processo envolve várias etapas, como a captura da imagem, a normalização da imagem, a extração de características (descritores) e, finalmente, a comparação dessas características com um banco de dados de impressões digitais previamente registradas.

As bibliotecas OpenCV e NumPy são amplamente utilizadas para realizar essas etapas. OpenCV, uma biblioteca de código aberto desenvolvida para visão computacional, fornece ferramentas para o processamento de imagens e a detecção de características. Já o NumPy facilita as operações matemáticas necessárias para o processamento e manipulação de dados em grandes volumes, como é o caso de imagens digitais.

3. MATERIAL E MÉTODOS

O sistema desenvolvido foi criado utilizando a linguagem de programação Python, que oferece recursos poderosos e bibliotecas como OpenCV, NumPy e Flask para o processamento de imagens e a criação da interface web. A primeira etapa do processo de autenticação envolve a captura da imagem da impressão digital. A imagem é então normalizada para facilitar a extração das características da impressão digital.

Uma vez que a imagem foi normalizada, o sistema utiliza técnicas de extração de características, como o cálculo de gradientes e frequências de ridges, para identificar padrões exclusivos das impressões digitais. Os descritores extraídos da imagem são então comparados com os armazenados no banco de dados, determinando se o usuário é autorizado ou não.

4. CÓDIGOS-FONTE

4.1 RUN.PY

O arquivo **run.py** contém a configuração e o gerenciamento do servidor web para o sistema de identificação e autenticação biométrica. Ele utiliza o framework Flask para criar as rotas da aplicação, permitindo que o usuário se autentique por meio de uma imagem de impressão digital. A aplicação usa o banco de dados SQLite para armazenar informações sobre os agricultores, com acesso controlado por níveis de permissão.

```
from flask import Flask, render_template, redirect, request, url_for, flash,
session
from flask_sqlalchemy import SQLAlchemy
import app as login_user
import os
import re
import secrets
from werkzeug.utils import secure_filename

# Configurações da aplicação
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///pesticides.db'
app.config['UPLOAD_FOLDER'] = 'database/uploaded'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'tif', 'gif',
'tiff'}
app.secret_key = secrets.token_urlsafe(24)
db = SQLAlchemy(app)

# Função para verificar se o arquivo tem uma extensão permitida
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in app.con-
fig['ALLOWED_EXTENSIONS']

# Rota principal
@app.route('/')
def index():
    return redirect('/login', code=302)

# Rota de login
@app.route('/login')
def login():
    username = session.get('name')
    level = session.get('level')

    # Se o usuário já estiver logado, redireciona para a página de farmers
    if username and level:
        return redirect('/farmers', code=302)
    else:
        return render_template('login.html')
```

```python
# Processa o login
@app.route('/login', methods=['POST'])
def login_post():
    file = request.files['file']
    filename = file.filename

    if file and allowed_file(filename):
        filename = secure_filename(file.filename)  # Nome seguro para o ar-
quivo
        image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)

        # Verifica se o diretório existe
        if not os.path.exists(app.config['UPLOAD_FOLDER']):
            os.makedirs(app.config['UPLOAD_FOLDER'])

        file.save(image_path)

        try:
            # Processa a autenticação
            user = login_user.main(image_path)

            if user:
                session['name'] = user['name']
                session['level'] = user['level']
                return redirect('/farmers')
            else:
                flash('Impressão digital incorreta ou não autorizada!')
                return redirect('/login', code=302)
        finally:
            # Garantir que a imagem seja removida após o processamento
            try:
                os.remove(image_path)
            except PermissionError as e:
                print(f"Erro ao tentar remover a imagem: {e}")
                # Se o erro persistir, podemos não remover a imagem neste mo-
mento, mas podemos tentar novamente ou fazer outra ação
    else:
        flash('Formato de arquivo inválido! Envie uma imagem PNG, TIF, GIF ou
JPG.')
        return redirect('/login', code=302)

# Rota de logout
@app.route('/logout', methods=['POST'])
def logout():
    session.pop('name', None)
    session.pop('level', None)
    return redirect('/login', code=302)

# Rota para a página de farmers
@app.route('/farmers/')
def farmers():
    username = session.get('name')
    level = session.get('level')

    # Verifica se o usuário está logado
    if username and level:
```

```
        levels = [*range(1, level + 1)]
        levels_string = str(levels).replace('[','(').replace(']',')')

        # Realiza a consulta no banco de dados
        results = db.engine.execute(f'SELECT * FROM farmers WHERE category in
{levels_string}')
        return render_template('farmers.html', results=results, username=user-
name, level=level)
    else:
        flash('Não autorizado!')
        return redirect('/login', code=302)

# Inicia o servidor web
if __name__ == "__main__":
    app.run(debug=True)
```

4.2 APP.PY

O arquivo **app.py** é responsável pela implementação das funcionalidades de processamento e comparação das impressões digitais para autenticação de usuários. Ele contém funções que lidam com o processamento da imagem de impressão digital, extração de descritores, e a comparação desses descritores com os armazenados de usuários autorizados.

```
import cv2
import os
import pickle
import numpy as np
from enhance.image_enhance import image_enhance
from skimage.morphology import skeletonize

# Função para remover pontos indesejados
def removedot(invertThin):
    temp0 = np.array(invertThin[:])
    temp1 = temp0 / 255
    temp2 = np.array(temp1)

    W, H = temp0.shape[:2]
    filtersize = 6

    for i in range(W - filtersize):
        for j in range(H - filtersize):
            filter0 = temp1[i:i + filtersize, j:j + filtersize]
            flag = 0
            if sum(filter0[:, 0]) == 0:
                flag += 1
            if sum(filter0[:, filtersize - 1]) == 0:
                flag += 1
            if sum(filter0[0, :]) == 0:
                flag += 1
            if sum(filter0[filtersize - 1, :]) == 0:
                flag += 1
            if flag > 3:
                temp2[i:i + filtersize, j:j + filtersize] = np.zeros((filter-
size, filtersize))
    return temp2

# Função para obter os descritores da imagem
def get_descriptors(img):
    img = image_enhance(img)  # Chama image_enhance que retorna uma imagem
processada

    # Convertendo a imagem para uint8
    img = np.array(img, dtype=np.uint8)

    print("Imagem processada para extração de descritores.")

    # Threshold da imagem
```

```
    ret, img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)
    img[img == 255] = 1

    # Thinning (afinamento)
    skeleton = skeletonize(img)
    skeleton = np.array(skeleton, dtype=np.uint8)
    skeleton = removedot(skeleton)

    # Detecção de cantos Harris
    harris_corners = cv2.cornerHarris(img, 3, 3, 0.04)
    harris_normalized = cv2.normalize(harris_corners, 0, 255,
norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32FC1)

    print("Harris corners calculados.")

    # Extração de keypoints usando os cantos Harris
    threshold_harris = 10
    keypoints = []
    for x in range(0, harris_normalized.shape[0]):
        for y in range(0, harris_normalized.shape[1]):
            if harris_normalized[x][y] > threshold_harris:
                keypoints.append(cv2.KeyPoint(y, x, 1))

    print(f"Keypoints encontrados: {len(keypoints)}")  # Verificação dos
keypoints encontrados

    # Criação do descritor ORB
    orb = cv2.ORB_create(nfeatures=1000)  # Aumenta o número de keypoints de-
tectados

    # Cálculo dos descritores
    _, des = orb.compute(img, keypoints)

    # Verificar se os descritores foram gerados corretamente
    if des is None:
        print("Erro: Nenhum descritor gerado!")
        return [], None  # Retorna uma lista vazia de keypoints e None para os
descritores
    else:
        print(f"Descritores gerados: {des.shape}")
        return keypoints, des

# Função para carregar ou gerar descritores para uma imagem
def get_des(image_path):
    if os.path.exists(image_path):
        img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Lê a imagem em
escala de cinza
        if img is None:
            raise ValueError(f"Erro ao carregar a imagem: {image_path}")
        return get_descriptors(img)
    else:
        raise FileNotFoundError(f"O arquivo {image_path} não existe.")

# Função principal para autenticação
def main(image_path):
```

```
    try:
        des = get_des(image_path)[1]  # Obtém os descritores da imagem forne-
cida

        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
        user = comparisons_with_permitted_images(des, bf)

        if user:
            return user
        else:
            return False
    except Exception as e:
        print(f"Erro ao tentar autenticar: {e}")
        return None

# Função para comparar descritores
def comparisons_with_permitted_images(sample_fingerprint, bf):
    score_threshold = 30
    print(f"Verificando impressões digitais autorizadas...")

    for user in users_authorization_and_authentication():
        try:
            permitted_fingerprint = get_des_permitted(user["fingerprint"])
            print(f"Descritores para {user['name']}: {permitted_fingerprint}")
            if permitted_fingerprint is None:
                continue  # Pula usuários cujo descritor não foi carregado
corretamente

            matches = bf.match(sample_fingerprint, permitted_fingerprint)
            matches = sorted(matches, key=lambda match: match.distance)
            print(f"Matches encontrados: {len(matches)}")

            if len(matches) > 0:
                score = sum([match.distance for match in matches]) /
len(matches)
                print(f"Score para {user['name']}: {score}")
            else:
                score = float('inf')

            if score < score_threshold:
                print(f"Usuário {user['name']} autenticado com sucesso!")
                return user
        except Exception as e:
            print(f"Erro ao comparar a impressão digital do usuário
{user['name']}: {e}")

    print("Nenhum usuário autenticado!")
    return None

# Função para carregar ou gerar descritores permitidos
def get_des_permitted(image_name):
    image_pickle = f"database/pickles/{image_name.split('.')[0]}.pkl"
    if os.path.exists(image_pickle):
        with open(image_pickle, 'rb') as pickle_file:
            return pickle.load(pickle_file)
    else:
```

```
        image_path = f"database/permitted/{image_name}"
        des = get_des(image_path)[1]
        with open(image_pickle, 'wb') as pickle_file:
            pickle.dump(des, pickle_file)
        return des

# Banco de usuários permitidos
def users_authorization_and_authentication():
    return [
        {"name": "101", "fingerprint": "101_1.tif", "level": 1},
        {"name": "102", "fingerprint": "102_1.tif", "level": 1},
        {"name": "103", "fingerprint": "103_1.tif", "level": 2},
        {"name": "104", "fingerprint": "104_1.tif", "level": 3},
    ]

# Teste para garantir que o arquivo está sendo executado corretamente
if __name__ == "__main__":
    test_path = "caminho/para/sua/imagem.tif"
    try:
        user = main(test_path)
        if user:
            print(f"Usuário autenticado: {user['name']} com nível {user['le-
vel']}")
        else:
            print("Nenhum usuário correspondente encontrado.")
    except Exception as e:
        print(f"Erro: {e}")
```

4.3 FREQUENCY.PY

O arquivo **frequency.py** contém a função frequency responsável por calcular a frequência espacial das linhas de uma impressão digital com base em sua imagem e na orientação das linhas. Essa função é fundamental para o processamento de imagens de impressões digitais, especialmente no contexto de autenticação biométrica.

```
import numpy as np
import math
import scipy.ndimage

def frequency(im, orientim, windsze, minWaveLength, maxWaveLength):
    """
    Função para calcular a frequência espacial das linhas de uma impressão di-
gital.

    Args:
        im : ndarray
            Imagem de entrada (impressão digital).
        orientim : ndarray
            Imagem com as orientações das linhas.
        windsze : int
            Tamanho da janela para a dilatação.
        minWaveLength : float
            Comprimento de onda mínimo permitido.
        maxWaveLength : float
            Comprimento de onda máximo permitido.

    Retorna:
        freqim : ndarray
            Imagem com a frequência das linhas.
    """
    rows, cols = np.shape(im)

    # Calcular a orientação média no bloco (evitar problemas de wraparound)
    cosorient = np.mean(np.cos(2 * orientim))
    sinorient = np.mean(np.sin(2 * orientim))
    orient = math.atan2(sinorient, cosorient) / 2

    # Rotaciona a imagem para que as linhas fiquem verticais
    rotim = scipy.ndimage.rotate(
        im,
        orient / np.pi * 180 + 90,
        axes=(1, 0),
        reshape=False,
        order=3,
        mode='nearest'
    )

    # Ajuste da imagem rotacionada para evitar regiões inválidas
    cropsze = int(np.fix(rows / np.sqrt(2)))
    offset = int(np.fix((rows - cropsze) / 2))
    rotim = rotim[offset:offset + cropsze][:, offset:offset + cropsze]
```

```
    # Projeção dos valores cinza nas colunas
    proj = np.sum(rotim, axis=0)
    dilation = scipy.ndimage.grey_dilation(proj, windsze, struc-
ture=np.ones(windsze))

    # Cálculo das diferenças entre a dilatação e a projeção original
    temp = np.abs(dilation - proj)

    # Limiar para detectar picos
    peak_thresh = 2
    maxpts = (temp < peak_thresh) & (proj > np.mean(proj))
    maxind = np.where(maxpts)

    rows_maxind, cols_maxind = np.shape(maxind)

    # Verifica se picos foram encontrados e calcula a frequência espacial
    if cols_maxind < 2:
        freqim = np.zeros(im.shape)  # Sem picos, sem frequência
    else:
        NoOfPeaks = cols_maxind
        waveLength = (maxind[0][cols_maxind - 1] - maxind[0][0]) / (NoOfPeaks
- 1)

        # Se o comprimento de onda estiver dentro do intervalo permitido
        if minWaveLength <= waveLength <= maxWaveLength:
            freqim = 1 / np.double(waveLength) * np.ones(im.shape)
        else:
            freqim = np.zeros(im.shape)  # Fora dos limites, define a frequên-
cia como 0

    return freqim
```

4.4 IMAGE_ENHANCE.PY

O arquivo image_enhance.py contém a função **image_enhance**, responsável pelo aprimoramento da imagem de uma impressão digital para extrair características relevantes, como orientação e frequência, que são cruciais para o processo de autenticação biométrica.

```
from .ridge_segmentation import ridge_seg  # Corrigido para ridge_seg
from .ridge_orientation import ridge_orient
from .ridge_frequency import ridge_freq
from .ridge_filter import ridge_filter

def image_enhance(img):
    """
    Função principal de aprimoramento de imagem. Realiza o processamento da
impressão digital
    para extrair características relevantes como orientação e frequência.

    Args:
        img : ndarray
            Imagem de entrada, que será processada.

    Retorna:
        newim : ndarray
            Imagem binarizada resultante do processamento de aprimoramento.
    """
    # Definindo os parâmetros para a segmentação
    blksze = 16  # Tamanho do bloco para segmentação
    thresh = 0.1  # Limiar para segmentação
    normim, mask = ridge_seg(img, thresh)  # Segmentação da imagem

    # Parâmetros para a orientação das linhas de impressão digital
    gradientsigma = 1
    blocksigma = 7
    orientsmoothsigma = 7
    # Calculando a orientação das linhas
    orientim = ridge_orient(normim, gradientsigma, blocksigma, orientsmoo-
thsigma)

    # Parâmetros para o cálculo da frequência
    blksze = 38
    windsze = 5
    minWaveLength = 5
    maxWaveLength = 15
    # Calculando a frequência das linhas de impressão digital
    freq, medfreq = ridge_freq(normim, mask, orientim, blksze, windsze, minWa-
veLength, maxWaveLength)

    freq = medfreq * mask  # Máscara aplicada para a frequência
    kx = 0.65  # Coeficiente para o filtro
    ky = 0.65  # Coeficiente para o filtro
    # Criação e aplicação do filtro Gabor para o aprimoramento da imagem
    newim = ridge_filter(normim, orientim, freq, kx, ky)
```

```
    # Retornando a imagem binarizada após o processamento
    return newim < -3
```

4.5 RIDGE_FILTER.PY

O arquivo **ridge_filter.py** contém a função ridge_filter, que aplica um filtro Gabor à imagem da impressão digital para aprimorar suas características, como as linhas das impressões digitais, com base nas orientações e frequências calculadas anteriormente. Este processo de filtragem ajuda a destacar padrões importantes, como as cristas e sulcos das impressões digitais, que são cruciais para a autenticação biométrica.

```
import numpy as np
import scipy.ndimage


def ridge_filter(im, orient, freq, kx, ky):
    angleInc = 3  # Incremento do ângulo para a rotação do filtro
    im = np.float64(im)  # Garantir que a imagem seja do tipo correto
    rows, cols = im.shape
    newim = np.zeros((rows, cols))

    # Flatten freq to 1D and get indices of non-zero frequencies
    freq_1d = np.reshape(freq, (1, rows * cols))
    ind = np.where(freq_1d > 0)

    ind = np.array(ind)[1, :]  # Extraímos os índices relevantes

    # Round the frequencies to 2 decimal points
    non_zero_elems_in_freq = np.round(freq_1d[0][ind] * 100) / 100.0
    unfreq = np.unique(non_zero_elems_in_freq)

    # Generate filters for these unique frequencies
    sigmax = 1 / unfreq[0] * kx
    sigmay = 1 / unfreq[0] * ky

    # Calculate the filter size
    sze = int(np.round(3 * max(sigmax, sigmay)))

    # Meshgrid to generate the filter
    x, y = np.meshgrid(np.linspace(-sze, sze, 2 * sze + 1), np.linspace(-sze,
sze, 2 * sze + 1))

    # Create the Gabor filter
    reffilter = np.exp(-((x**2 / (sigmax**2)) + (y**2 / (sigmay**2)))) *
np.cos(2 * np.pi * unfreq[0] * x)

    filt_rows, filt_cols = reffilter.shape

    # Initialize a filter bank for different orientations
    gabor_filter = np.zeros((int(180 / angleInc), int(filt_rows),
int(filt_cols)))

    for o in range(0, int(180 / angleInc)):
```

```
        rot_filt = scipy.ndimage.rotate(reffilter, -(o * angleInc + 90),
reshape=False)
        gabor_filter[o] = rot_filt

    # Find valid indices inside the image boundaries
    maxsze = int(sze)
    temp = freq > 0
    validr, validc = np.where(temp)

    # Ensure valid indices are within the bounds
    validr = validr[(validr > maxsze) & (validr < rows - maxsze)]
    validc = validc[(validc > maxsze) & (validc < cols - maxsze)]

    # Convert orientation matrix from radians to index
    maxorientindex = int(np.round(180 / angleInc))
    orientindex = np.round(orient / np.pi * 180 / angleInc).astype(int)

    # Apply the Gabor filter to the image
    for r, c in zip(validr, validc):
        img_block = im[r - sze:r + sze + 1, c - sze:c + sze + 1]
        idx = orientindex[r, c] % maxorientindex  # Ensure that the index is
within range
        newim[r, c] = np.sum(img_block * gabor_filter[idx])

    return newim
```

4.6 RIDGE_FREQUENCY.PY

O arquivo ridge_freq.py contém a função ridge_freq, que é responsável por calcular a frequência das linhas de uma impressão digital, considerando a orientação das linhas e outras características da imagem. A função utiliza a função frequency do arquivo frequency.py para calcular a frequência em blocos específicos da imagem.

```
import numpy as np
from .frequency import frequency  # Importa a função de cálculo de frequência

def ridge_freq(im, mask, orient, blksze, windsze, minWaveLength, maxWaveLen-
gth):
    rows, cols = im.shape
    freq = np.zeros((rows, cols))  # Inicializa a matriz de frequências

    # Loop sobre a imagem em blocos de tamanho blksze
    for r in range(0, rows - blksze, blksze):
        for c in range(0, cols - blksze, blksze):
            # Extrai o bloco de imagem e o bloco de orientação
            blkim = im[r:r + blksze, c:c + blksze]
            blkor = orient[r:r + blksze, c:c + blksze]

            # Calcula a frequência para o bloco e atribui à matriz de frequên-
cias
            freq[r:r + blksze, c:c + blksze] = frequency(blkim, blkor, win-
dsze, minWaveLength, maxWaveLength)

    # Aplica a máscara na matriz de frequências
    freq = freq * mask

    # Flatten a matriz freq para 1D e encontra os elementos não nulos
    non_zero_elems_in_freq = freq[freq > 0]

    # Cálculo da média e mediana das frequências não nulas
    meanfreq = np.mean(non_zero_elems_in_freq)  # Média das frequências
    medianfreq = np.median(non_zero_elems_in_freq)  # Mediana das frequências

    return freq, medianfreq
```

4.7 RIDGE_ORIENTATION.PY

O arquivo ridge_orient.py contém a função **ridge_orient.py**, responsável por calcular a orientação das linhas de uma impressão digital com base na imagem de entrada. A orientação das linhas é um passo crucial para o processamento de imagens de impressões digitais, pois ajuda a identificar a direção das cristas e sulcos, facilitando o reconhecimento e a autenticação biométrica.

```
import numpy as np
import cv2
from scipy import ndimage
from scipy import signal

def ridge_orient(im, gradientsigma, blocksigma, orientsmoothsigma):
    rows, cols = im.shape

    # Calcular o tamanho do filtro gaussiano baseado em gradientsigma
    sze = int(6 * gradientsigma)
    if sze % 2 == 0:
        sze += 1  # Garantir que o tamanho do filtro seja ímpar

    # Criar o filtro gaussiano
    gauss = cv2.getGaussianKernel(sze, gradientsigma)
    f = gauss * gauss.T

    # Gradientes do filtro Gaussiano
    fy, fx = np.gradient(f)  # Gradient of Gaussian

    # Calcular os gradientes da imagem
    Gx = signal.convolve2d(im, fx, mode='same')
    Gy = signal.convolve2d(im, fy, mode='same')

    # Calcular a covariância dos gradientes
    Gxx = np.power(Gx, 2)
    Gyy = np.power(Gy, 2)
    Gxy = Gx * Gy

    # Suavizar as covariâncias para calcular a direção principal
    sze = int(6 * blocksigma)
    gauss = cv2.getGaussianKernel(sze, blocksigma)
    f = gauss * gauss.T

    # Convolução das covariâncias com o filtro Gaussiano
    Gxx = ndimage.convolve(Gxx, f)
    Gyy = ndimage.convolve(Gyy, f)
    Gxy = 2 * ndimage.convolve(Gxy, f)

    # Solução analítica para a direção principal
    denom = np.sqrt(np.power(Gxy, 2) + np.power((Gxx - Gyy), 2)) +
np.finfo(float).eps

    # Calcular os senos e cossenos dos ângulos duplicados
```

```
    sin2theta = Gxy / denom
    cos2theta = (Gxx - Gyy) / denom

    # Suavizar a orientação
    if orientsmoothsigma:
        sze = int(6 * orientsmoothsigma)
        if sze % 2 == 0:
            sze += 1  # Garantir que o tamanho do filtro seja ímpar
        gauss = cv2.getGaussianKernel(sze, orientsmoothsigma)
        f = gauss * gauss.T
        cos2theta = ndimage.convolve(cos2theta, f)
        sin2theta = ndimage.convolve(sin2theta, f)

    # Calcular a orientação final
    orientim = np.pi / 2 + np.arctan2(sin2theta, cos2theta) / 2
    return orientim
```

4.8 RIDGE_SEGMENTATION.PY

O arquivo **ridge_seg.py** contém a função **ridge_seg**, responsável por realizar a segmentação das linhas de uma impressão digital com base na intensidade dos gradientes da imagem. A segmentação é uma etapa crucial para isolar as características importantes das impressões digitais e para o processo de autenticação biométrica.

```
import numpy as np
import cv2

# Defina o BLOCKSIZE, caso contrário, ele gerará um erro.
BLOCKSIZE = 16  # Exemplo de valor, ajuste conforme necessário

def ridge_seg(im, threshold=0.5):
    """
    Função para segmentar as linhas de uma impressão digital com base no li-
miar de intensidade.

    Args:
    im : ndarray
        Imagem de entrada (impressão digital).
    threshold : float, opcional
        Limiar para a segmentação da imagem, o valor padrão é 0.5.

    Retorna:
    normim : ndarray
        Imagem normalizada.
    segmented_image : ndarray
        Imagem segmentada.
    """
    # Normaliza a imagem para a faixa [0, 1]
    im_norm = np.float32(im) / 255.0

    # Aplica um filtro de mediana para reduzir o ruído
    im_filtered = cv2.medianBlur(im_norm, 3)

    # Calcula o gradiente da imagem
    grad_x = cv2.Sobel(im_filtered, cv2.CV_64F, 1, 0, ksize=5)
    grad_y = cv2.Sobel(im_filtered, cv2.CV_64F, 0, 1, ksize=5)

    # Calcula a magnitude do gradiente
    magnitude = np.sqrt(grad_x**2 + grad_y**2)

    # Normaliza a magnitude para a faixa [0, 1]
    magnitude = np.uint8(255 * magnitude / np.max(magnitude))

    # Segmenta a imagem com base no limiar
    _, segmented_image = cv2.threshold(magnitude, threshold * 255, 255,
cv2.THRESH_BINARY)

    # Calcula o número de linhas e a altura da imagem
    rows, cols = segmented_image.shape[:2]
```

```
    new_rows = int(BLOCKSIZE * np.ceil(float(rows) / float(BLOCKSIZE)))  # Usando 'float' em vez de np.float

    # Aqui você pode continuar com a implementação da segmentação (por exemplo, calculando a máscara)
    # Implementação adicional conforme necessário

    return im_norm, segmented_image
```

5 CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema de **identificação e autenticação biométrica** utilizando impressões digitais foi um passo importante para o entendimento das técnicas de **processamento de imagem** e sua aplicação em **segurança de dados**. Ao longo deste trabalho, foram abordados os principais desafios enfrentados no processamento de **imagens biométricas**, como a segmentação das linhas das impressões digitais, a extração de características e a comparação com descritores de usuários previamente registrados.

A solução proposta, implementada em Python, fez uso de bibliotecas essenciais como **OpenCV**, **NumPy**, e **Flask** para realizar a manipulação e autenticação das imagens. O sistema foi capaz de validar a autenticidade das impressões digitais e fornecer diferentes níveis de acesso conforme a autenticação dos usuários, garantindo assim a segurança das informações.

Apesar de termos alcançado os objetivos propostos, a pesquisa trouxe à tona diversas oportunidades de melhoria. Uma possível extensão do projeto seria a utilização de **técnicas mais avançadas de aprendizado de máquina** e **inteligência artificial**, como redes neurais convulsionais, para aprimorar ainda mais o processo de **reconhecimento de padrões** em impressões digitais. Além disso, a **precisão e a escalabilidade** do sistema poderiam ser melhorados com a implementação de **algoritmos de aprendizado profundo** que permitam o processamento em tempo real e a integração com outras formas de biometria, como o reconhecimento facial ou de íris.

A implementação do sistema também abre caminho para futuras pesquisas sobre a **integração de sistemas biométricos com banco de dados sensíveis**, como os utilizados pelo **Ministério do Meio Ambiente**, no qual a autenticidade do usuário é crucial para o controle de informações estratégicas. Assim, o presente trabalho contribui para o avanço na área de **segurança biométrica**, propondo soluções viáveis para autenticação em ambientes críticos.

Por fim, espera-se que este estudo seja uma base para novos pesquisadores e desenvolvedores que desejam aprofundar o estudo da biometria digital e expandir suas

aplicações para outros domínios, como **segurança pública**, **saúde**, **finanças**, entre outros. As inovações e melhorias contínuas nesta área são essenciais para a construção de sistemas mais **seguros e eficientes**, promovendo um avanço significativo nas tecnologias de **autenticação e proteção de dados pessoais**.

REFERÊNCIAS

1. WIKIPEDIA. *OpenCV*. Disponível em: https://pt.wikipedia.org/wiki/OpenCV. Acesso em: 20 nov. 2024.

2. PYPI. *OpenCV-Python*. Disponível em: https://pypi.org/project/opencv-python/. Acesso em: 20 nov. 2024.

3. GITHUB. *OpenCV GitHub Repository*. Disponível em: https://github.com/opencv/opencv. Acesso em: 20 nov. 2024.

4. MEDIUM. *Face Recognition Using Python*. Disponível em: https://medium.com/mlcrunch/face-detection-using-dlib-hog-198414837945. Acesso em: 20 nov. 2024.

5. REALPYTHON. *How to Work with Images in Python using OpenCV*. Disponível em: https://realpython.com/tutorials/opencv/. Acesso em: 20 nov. 2024.

7. GITHUB. *Face Recognition*. Disponível em: https://github.com/ageitgey/face_recognition. Acesso em: 20 nov. 2024.

8. STACKOVERFLOW. *How to handle image files in Flask*. Disponível em: https://stackoverflow.com/questions/42668792/handle-image-files-in-flask. Acesso em: 20 nov. 2024.

9. PYPI. *Face Recognition Library*. Disponível em: https://pypi.org/project/face_recognition/. Acesso em: 20 nov. 2024.

10. TENSORFLOW. *Image Classification with TensorFlow*. Disponível em: https://www.tensorflow.org/tutorials/images/classification. Acesso em: 20 nov. 2024.

11. BLOG, MYSITE. *Using Python for Digital Image Processing*. Disponível em: https://www.mysite.com/image-processing. Acesso em: 20 nov. 2024.

12. DATACAMP. *Introduction to Image Processing with Python*. Disponível em: https://www.datacamp.com/community/tutorials/image-processing-python. Acesso em: 20 nov. 2024.