

Reconhecimento de Iris

Universidade Federal de Uberlândia (UFU)
Bacharelado em Sistemas de Informação (BSI)
Processamento Digital de Imagens (PDI)

11721BSI234 – Carlos Cabral de Menezes

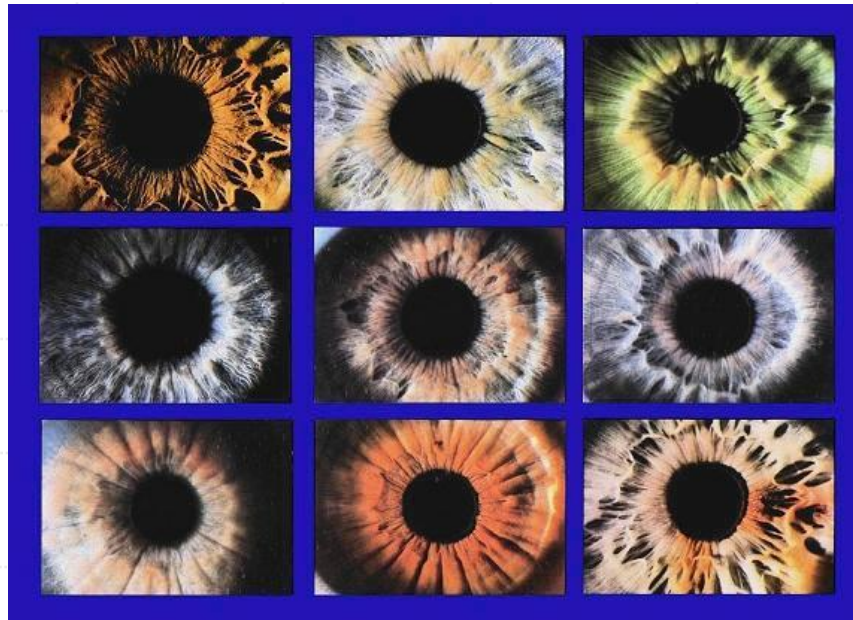
11811BSI221 – Gabriel Dal Belo Gomes Santos

11811BSI231 – Guilherme Bartasson Naves Junker

12121BCC015 – Guilherme Cabral de Menezes

Contexto

- A íris ganha sua forma aleatoriamente no período de gestação, sendo única por indivíduo e eficaz na identificação até mesmo de cegos e gêmeos.
- Sua detecção e reconhecimento são rápidos, compondo o sistema de identificação biométrica mais eficaz .



Dificuldades

- Converter a imagem em dados de forma que possa ser analisada e comparada sem que suas características sejam perdidas.
- Localizada em uma superfície úmida, curva e com reflexos. Cílios e pálpebras dificultam a identificação.
- Doenças também podem comprometer a identificação ao alterar a forma original da íris.

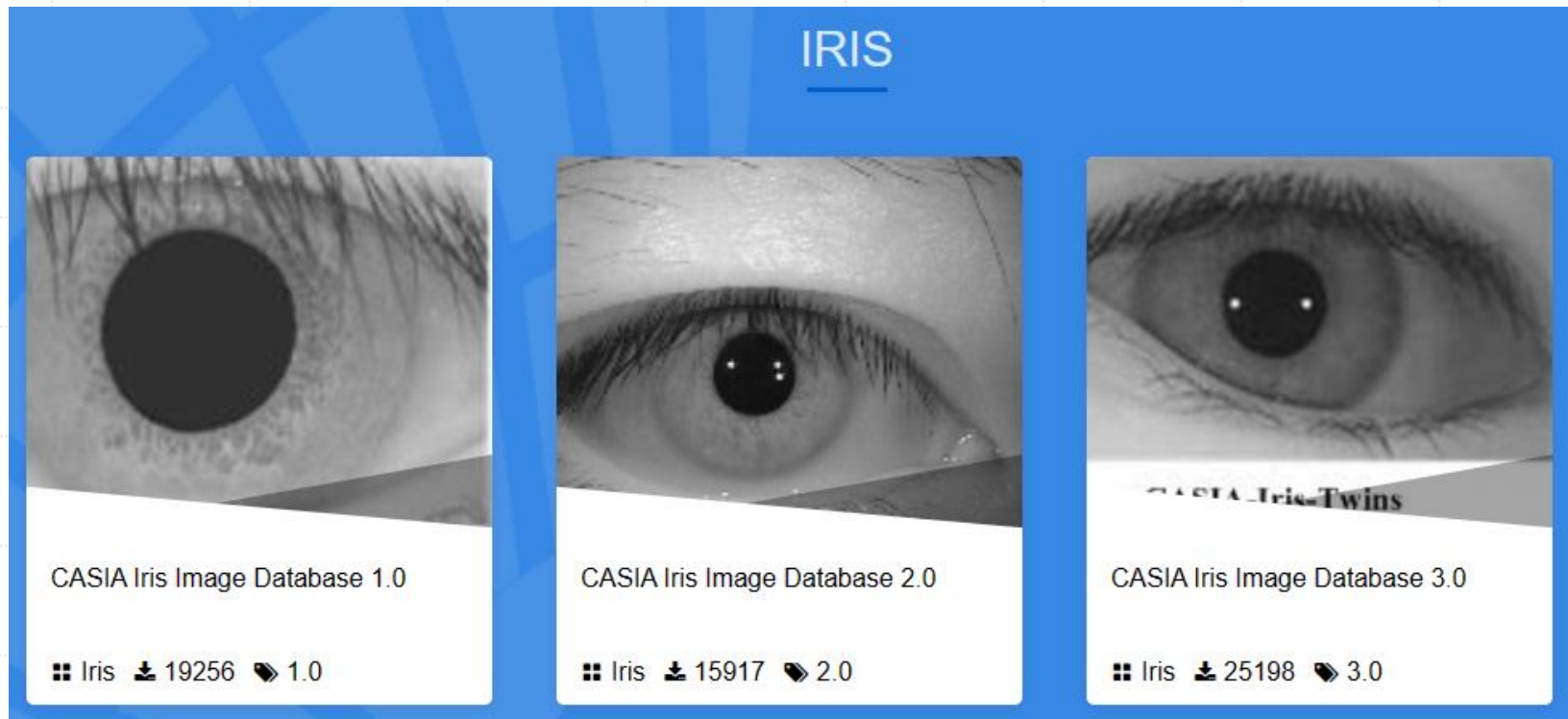
Descrição



- A proposta deste trabalho era a de compreender e replicar um sistema básico de reconhecimento e comparação de íris, utilizando a linguagem de programação Python e todas as ferramentas apresentadas na disciplina de Processamento Digital de Imagens.
- O trabalho utilizado como base para replicação foi encontrado no GitHub, e pertence ao usuário Thuy Ng ([thuyngch](#)), que o implementou tanto em Python quanto em Matlab.


Base de Dados


Todas as imagens utilizadas fazem parte do banco de dados online CASIA-IrisV1.
Nele, estão cadastradas 108 pessoas, cada uma possuindo 7 imagens de seus olhos.





Estrutura do Programa


- O código principal foi implementado no arquivo `verify.py`, que utiliza dos seguintes arquivos e funções auxiliares, presentes no folder Functions.


 `boundary.py`

 `createAccount.py`


 `encode.py`

 `extractFeature.py`

 `line.py`

 `matching.py`

 `normalize.py`

 `segment.py`

Camada 1

verify

Camada 2

matching

extractFeature

Camada 3

segment

normalize

encode

Camada 4

line

boundary





Camada 1 - verify


- Utiliza duas funções, que recebem como parâmetro os dados passados via linha de comando:
- **ExtractFeature** recebe o caminho até o arquivo que se deseja verificar (imagem de íris), e retorna o próprio arquivo, o caminho para o diretório que contém os templates à serem comparados, e a máscara de comparação - baseada em um determinado limiar também passado como parâmetro.
- **Matching** recebe os resultados da anterior, e busca em meio as demais a íris a ser verificada. Se o resultado for -1, é porque não há amostras registradas. Se for 0, é porque não há amostras compatíveis. Se não, mostra todas as amostras compatíveis em ordem decrescente de confiabilidade.



Camada 2 – extractFeature

É composta por três funções:

- **Segment**, que busca fragmentar a imagem em regiões de íris, pupila, e ruído.
- **Normalize**, que normaliza a imagem e retorna as regiões normalizadas da íris e de ruído.
- **Encode**, gera o template biométrico da íris e a máscara de ruídos partindo da região normalizada da própria íris.



Camada 3 – segment

Utiliza das funções implementadas em **Boundary** para detecção dos contornos da íris, e das funções em **Line** para detecção das regiões do topo e da base da pálpebra, detectáveis através de um valor limiar passado como parâmetro.

Recebe a imagem do olho, o limiar referente aos cílios, e uma flag para saber se multiprocessamento será usado ou não.

Retorna as coordenadas centralizadas e o raio do contorno tanto da íris quanto da pupila.



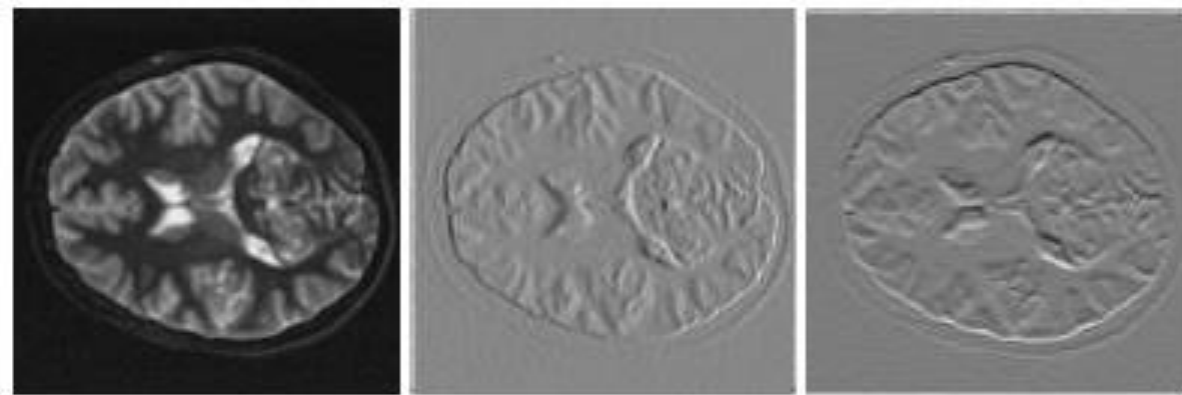
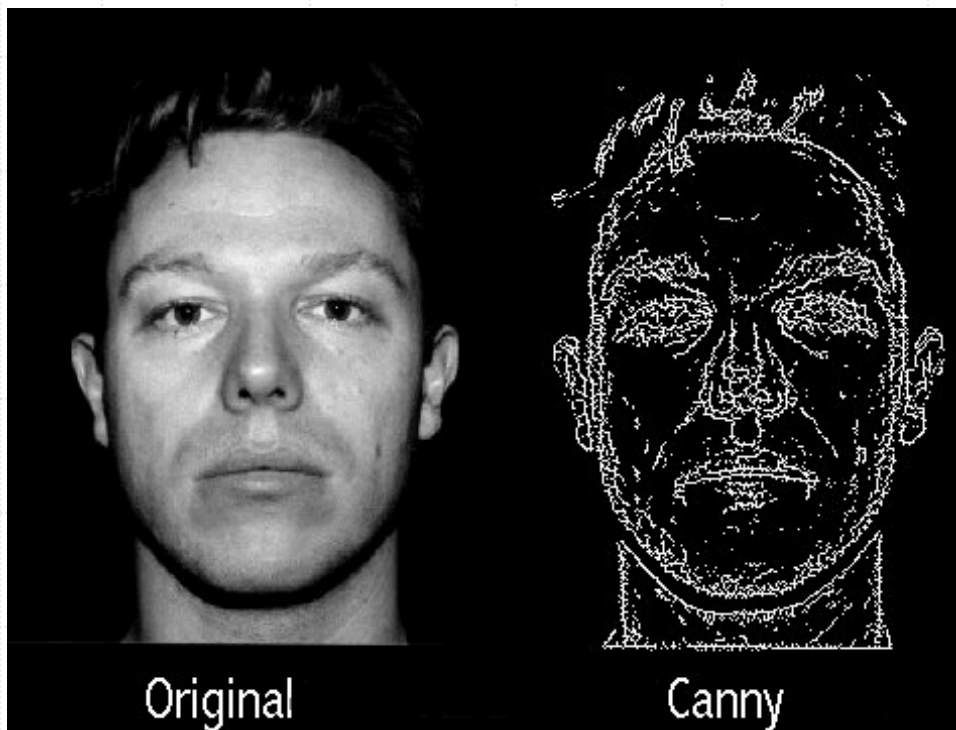
Camada 4 - line

Encontra as linhas em uma imagem. Para isso, foram utilizadas:

- A detecção de bordas de Canny,
- A transformada linear de Hough,
- A transformada de Radon.

Detecção de bordas de Canny

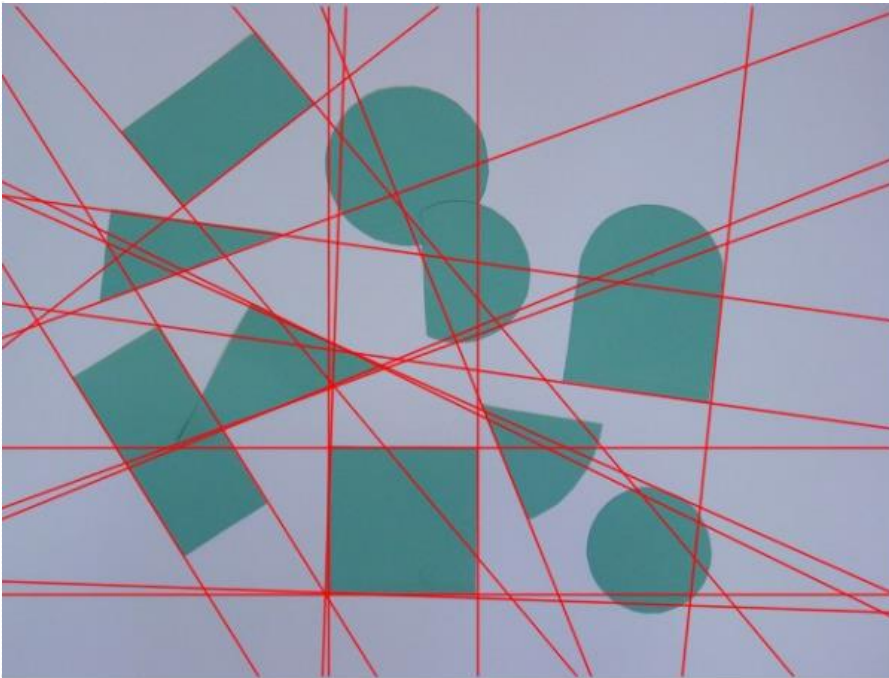
O filtro de Canny é um filtro de convolução que usa a primeira derivada. Ele suaviza o ruído e localiza bordas, combinando um operador diferencial com um filtro Gaussiano.



A transformada linear de Hough

Método padrão para detecção de formas que são facilmente parametrizadas (linhas, círculos, elipses, etc.) em imagens computacionais.

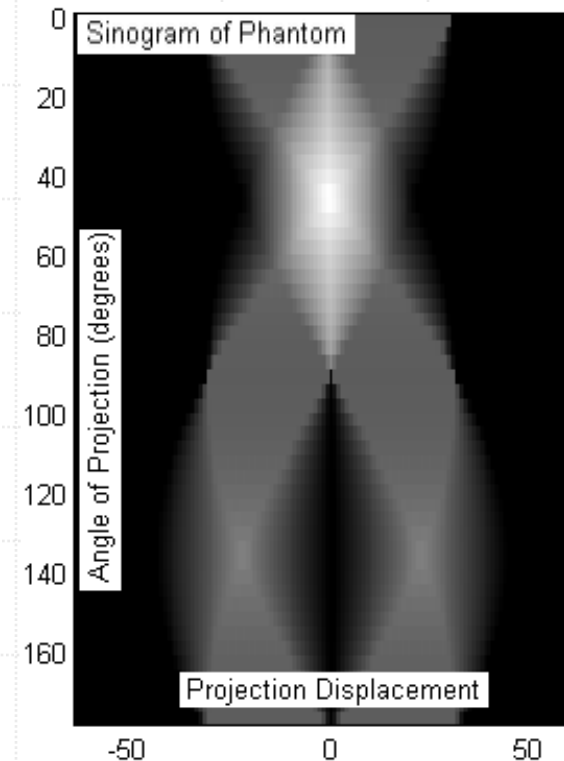
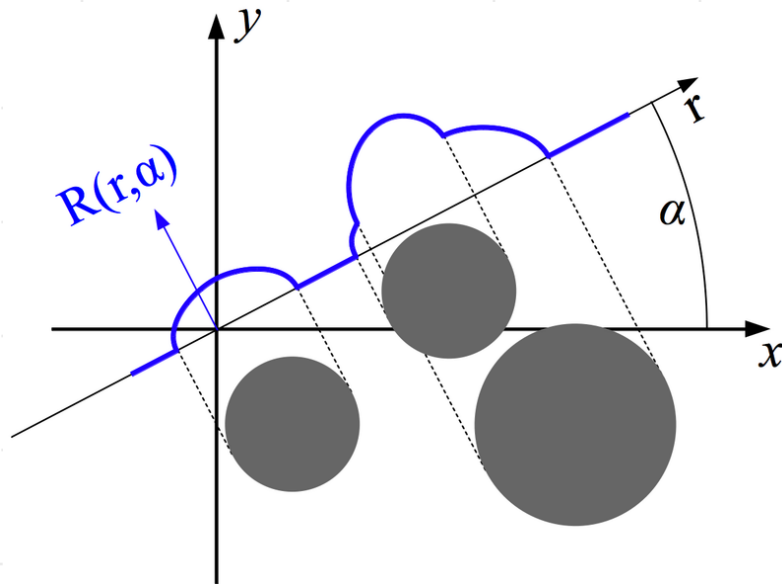
Em geral, a transformada é aplicada após a imagem sofrer um pré-processamento, como a detecção de bordas.



A transformada de Radon

Auxilia na detecção da estrutura interna de um objeto com base em seu contorno e na reconstrução de imagens através de projeções sobre linhas retas.

Muito usada no âmbito do diagnóstico por imagem (ultrassom, tomografia, e etc).



Camada 4 – boundary

Como o nome sugere, é responsável pela identificação dos contornos. Possui uma função para identificação do contorno interno da íris (`searchInnerBound`) e outra para identificação do contorno externo (`searchOuterBound`).

Ambas são semelhantes, e utilizam do espaço de Hough, de sua derivada parcial, e de uma terceira função que efetua os contornos de fato (`ContourIntegralCircular`).

Assim como no algoritmo de detecção de bordas de Canny, suavizar a imagem melhora o processamento. Aqui, utilizou-se a FFT para tal, antes de encontrar os contornos do olho.



Camada 3 - normalize

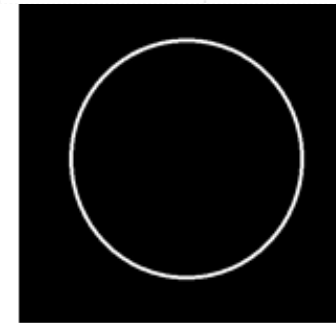
Deforma a região circular da iris de tal forma que ela se transforma em um bloco retangular de dimensões constantes já pré-definidas.

Essa função não utiliza nenhum algoritmo específico, apenas alguns conceitos geométricos, como por exemplo, o cosseno, o produto escalar e coordenadas polares.

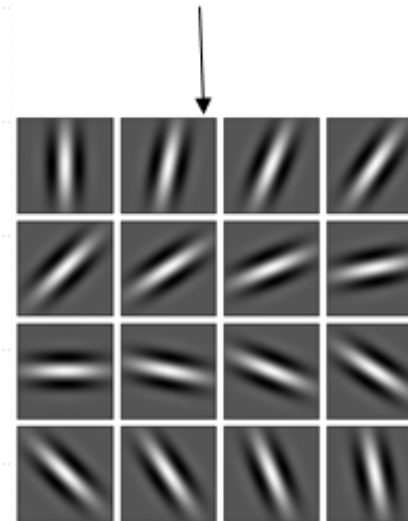
Camada 3 – encode

A função *encode* serve para gerar o template biométrico da íris e a máscara de ruído a partir da região normalizada retornada pela função *normalize*.

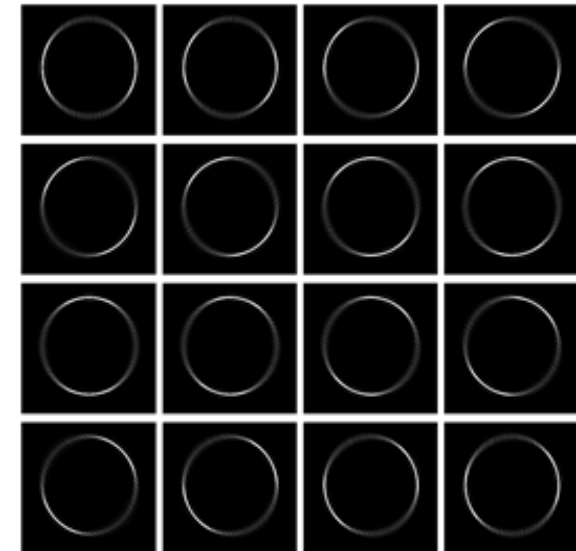
Para conseguir gerar o template biométrico foi necessário convulação (via FFT) com filtros de Gabor, como mostrado na imagem ao lado. Além disso foi necessário uma quantização, vendo que, nesse caso, o retorno da convulação foram coordenadas complexas.



Input Image of
a circle



A bank of 16 Gabor Filters



The output circle as seen when pass
through individual Gabor filter



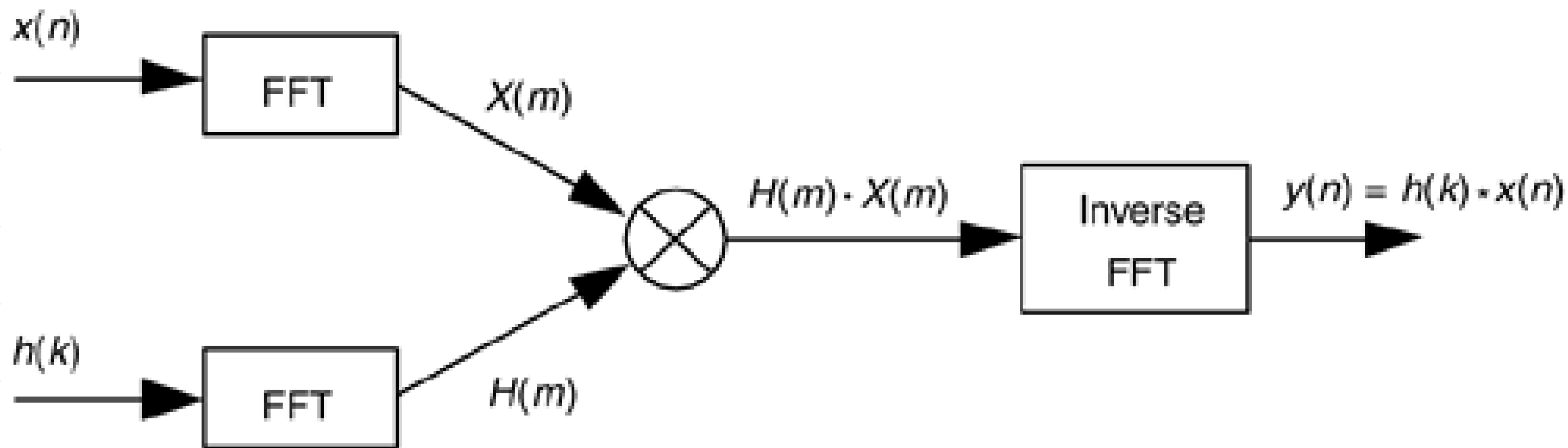
FFT

Teorema da Convolução

$$f(x, y) * g(x, y) \Leftrightarrow F(u, v)G(u, v)$$

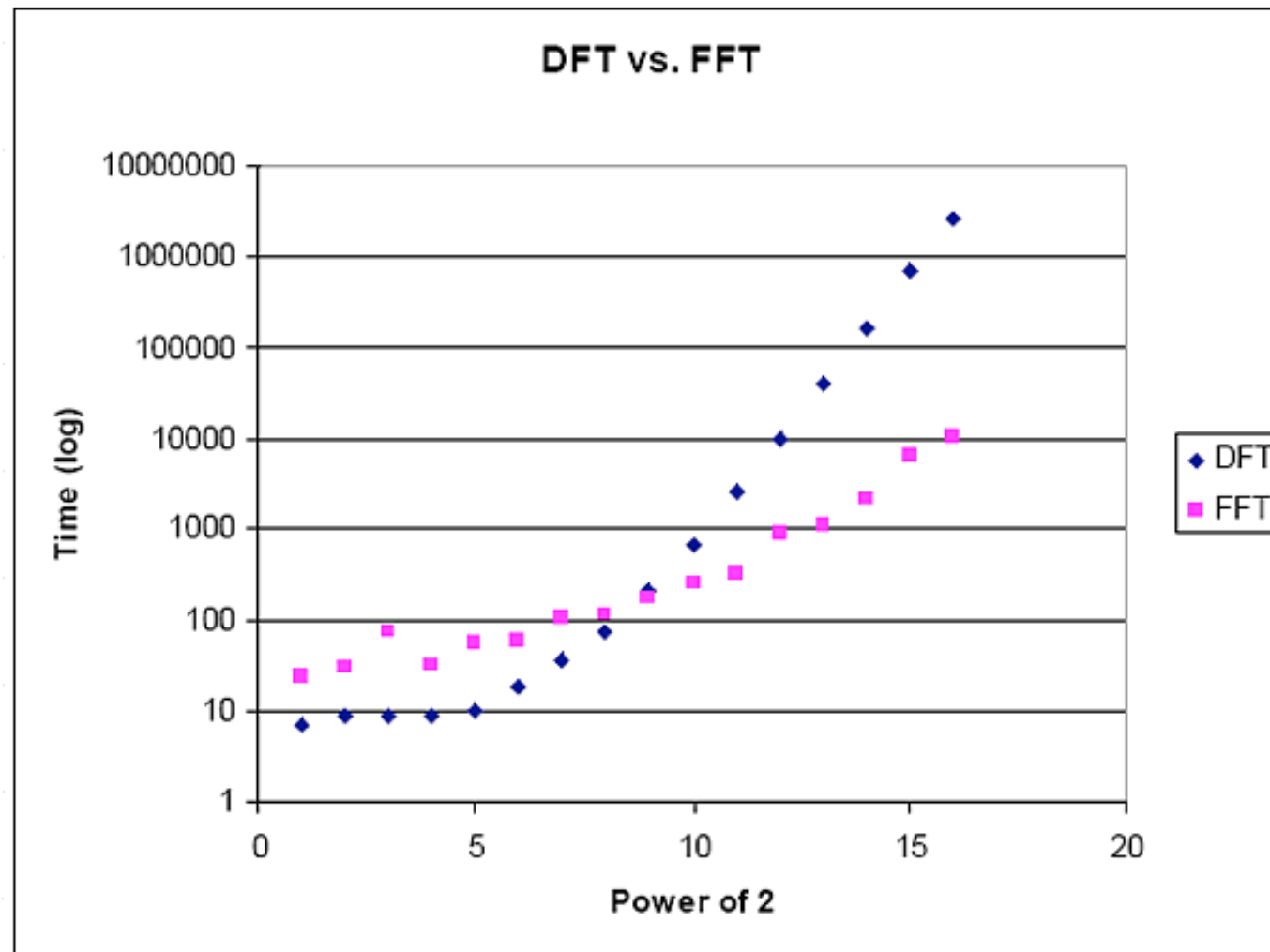
$$f(x, y)g(x, y) \Leftrightarrow F(u, v) * G(u, v)$$

FFT



Processo de convolução via FFT

FFT



DFT vs FFT



Camada 2 - matching

Retornando à segunda camada, neste ponto do código, o processamento da imagem já foi todo realizado. Basta fazer o *matching* da íris processada com o banco de dados online CASIA-IrisV1.

Para realizar o *matching*, foi utilizada a distância de Hamming para saber o grau de semelhança entre o template e o conjunto de imagens do banco de dados de uma íris.

Execução

- Não conseguimos executar o código original, devido à atualizações feitas pelo criador que não foram incluídas e explicadas no README.
- Traduzimos o código original todo, para fins de entendimento.
- Tentamos replicar a lógica e os resultados esperados no Jupyter.

```
mirror_mod = modifier_ob.  
# Add mirror object to mirror  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True
```

```
#selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.name))  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly
```

--- OPERATOR CLASSES ---

```
types.Operator):  
on X mirror to the selected  
object.mirror_mirror_x"  
mirror X"
```

```
context):  
context.active_object is not
```




Ferramentas e Técnicas

Conhecidas

- FFT
- Quantização
- Suavização Gaussiana

Novas

- Espaço de Hough
- Detecção de bordas de Canny
- A transformada de Radon

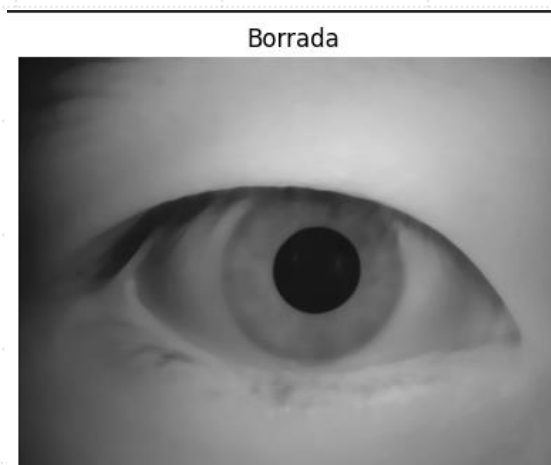
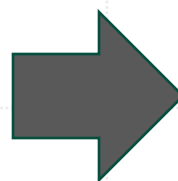
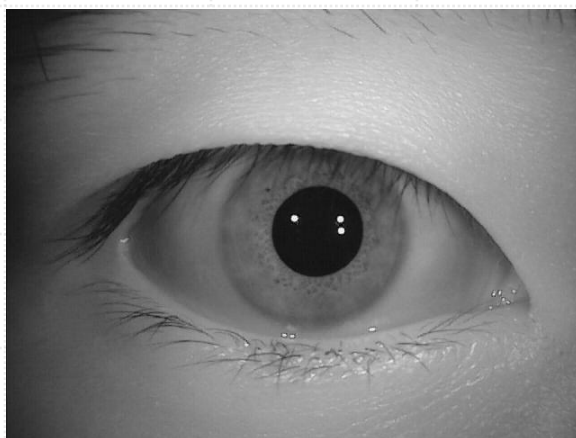


Extra

Originalmente, tentamos implementar nossa própria versão de um sistema identificador de íris. Porém, o mesmo apenas identificava o ponto central da pupila e supunha o raio da íris com base no tamanho da imagem.

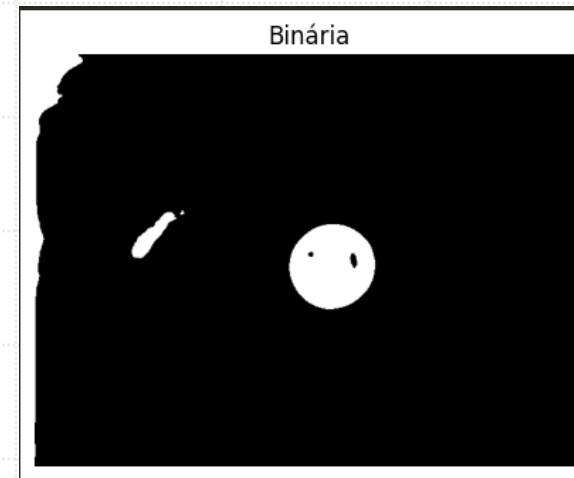
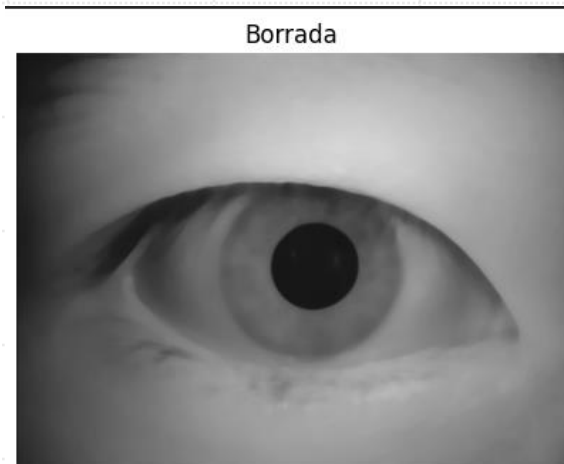
- Inicialmente borramos a imagem a fim de reduzir ruídos e remover partes indesejadas como os cílios, por exemplo.

```
img = cv2.imread("S2002R01.jpg", cv2.IMREAD_GRAYSCALE)
h, w = img.shape
radius = int(h * .095)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (radius, radius))
img = cv2.medianBlur(img, 13)
```



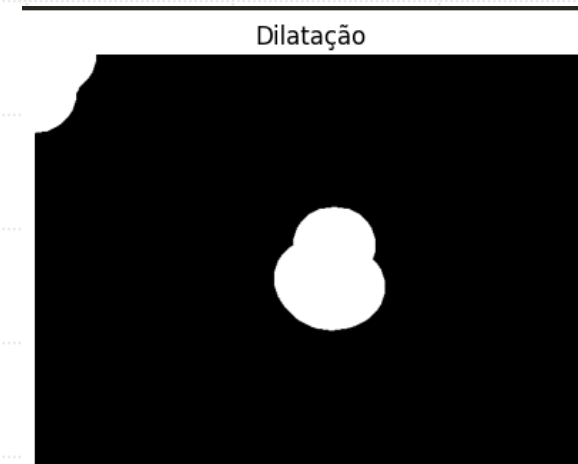
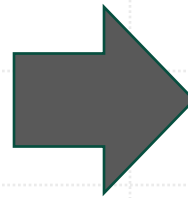
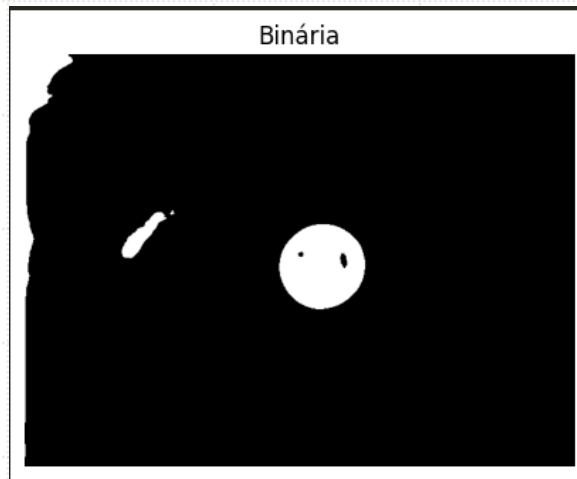
- Em seguida iremos binarizar a imagem

```
ret, binary = cv2.threshold(img, 30, 255, cv2.THRESH_BINARY_INV)
```



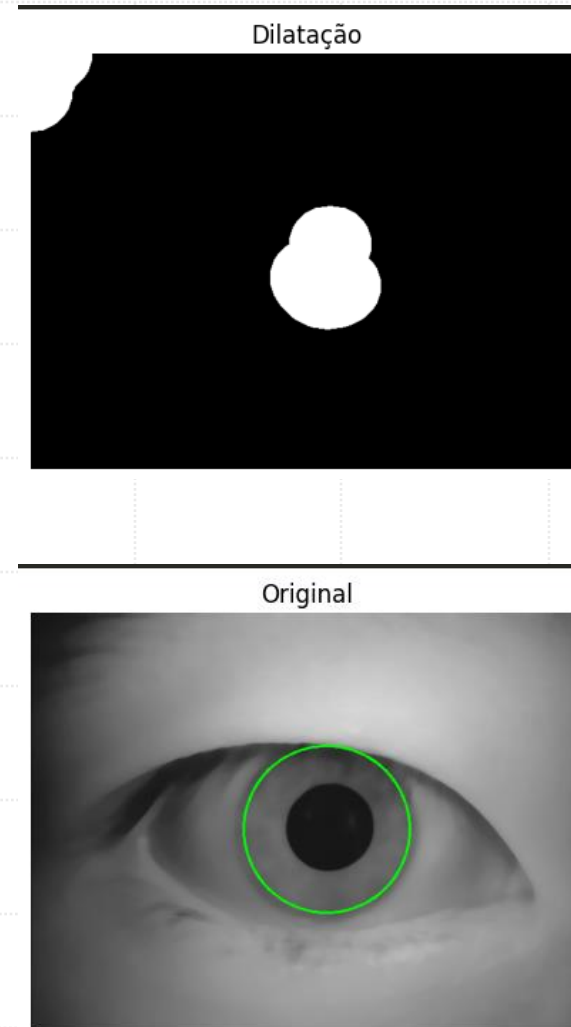
- Realizaremos um procedimento de abertura seguido de uma dilatação para remover elementos que não fazem parte da íris

```
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)  
dilate = cv2.morphologyEx(opening, cv2.MORPH_DILATE, kernel)
```



- Por fim obteremos os contornos a fim de encontrar o centro da íris e a partir deste centro desenhar um círculo baseado no tamanho da imagem que possivelmente irá englobar a íris

```
contours, hierarchy =  
cv2.findContours(dilate, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  
  
if len(contours) != 0:  
    cnt = contours[0]  
    M1 = cv2.moments(cnt)  
  
    Cx1 = int(M1['m10'] / M1['m00'])  
    Cy1 = int(M1['m01'] / M1['m00'])  
  
    center1 = (int(Cx1), int(Cy1))  
  
    cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)  
    cv2.circle(cimg, center1, int(h * .2), (0, 255, 0), 2)
```





Referências

Hough: [hough.pdf \(usp.br\)](#)

Radon: [Transformada de Radon – Wikipédia, a enciclopédia livre \(wikipedia.org\)](#)

Canny: [Untitled Document \(uff.br\)](#)

FFT: [Teorema da Convolução](#)