Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Pernambuco

Relatório Técnico

Disciplina: Visão Computacional - 2017.2
Prof.: Dr. Carlos Alexandre Barros de Mello

Aluno: Francisco de Assis de Souza Rodrigues

Artigo selecionado: https://doi.org/10.1016/S0167-8655(00)00008-8
Título: *A new face detection method based on shape information*
Autores: *JianguoWang; TieniuTan.*
Publicação: *Pattern Recognition Letters*
Ano: 2000

1. Introdução

A detecção de facial é o primeiro passo importante no processo de reconhecimento facial. O artigo dos autores *Jianguo Wang* e *Tieniu Tan [1]* abordam esse tema. Sobretudo, detecção facial pode ser considerado como um problema de segmentação de imagem, pois a imagem deve ser segmentada em duas partes: uma contendo faces, e outra, regiões não visíveis.

Existem inúmeros métodos para extrair caracteristicas faciais, entretanto, com a percepção dos autores, que as fotos com fundo simples, são geralmente utilizadas em aplicações de *face-id*, para identificação de pessoas, e trabalho de segurança. am é notado que muitas características do contorno das cabeças humanas nessas imagens, são semelhantes a uma elipse, porém não é possível facilmente modelar usando apenas uma elipse.

Os autores mostram um modelo estendido do método de forma elíptica, aplicando um método que adiciona informações das bordas para detecção, onde seus experimentos inferem que o novo sistema eficiente no processamento de imagens com um fundo simples, independentemente de variações no tamanho, pose principal (rotação da cabeça moderada) e condição de iluminação.

1.1 **Motivação**

A detecção automática de rosto humano é um dos problemas mais difíceis na área de reconhecimento de padrões, afetando diretamente a mesma no processo de

reconhecimento facial. Sendo assim, existem grandes desafios a serem solucionados e aperfeiçoados em diversos aspectos.

1.2 Objetivos

O objetivo do trabalho de *JianguoWang; TieniuTan: A new face detection method based on shape information [1]*, é expandir o método de detecção de face por extração de características de forma, para detectar face em *backgroud* simples, com base em direção de bordas e padrão anel elíptico. Assim o meu trabalho, tem a finalidade de verificar, implementar, testar e comparar os resultados do método proposto pelos autores.

2. Metodologia

O problema de não poder representar uma face com apenas uma elipse, foi tratado pelos autores subdivididos em cinco partes, sendo os dois últimos passos, são o núcleo do modelo dos autores, como mostra o fluxo da imagem da Figura 01.

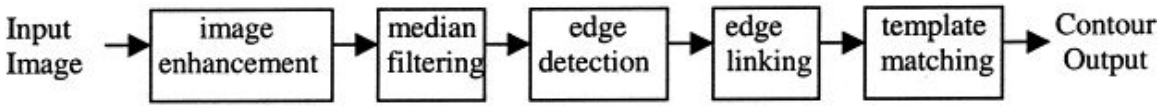

Figura 01: Fluxo do modelo de detecção de face.
Fonte: [1]

1º parte - melhoria da imagem

Nesta etapa, a imagem de entrada pode ter um contraste muito fraco, devido às condições de iluminação, então, é realizada uma equalização de histograma na mesma, a fim de melhorar o contraste da imagem original [1]. Esse método é útil para imagens brilhantes ou escuras, através desse ajuste, as intensidades podem ser melhor distribuídas no histograma, permitindo que as áreas de menor contraste local, ganhe um contraste maior, expandindo efetivamente os valores de intensidade mais frequentes [2].

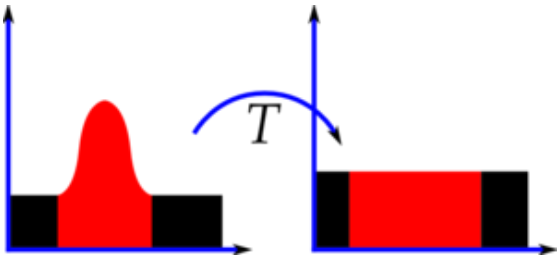

Figura 02: Histogram de equalização de uma imagem.
Fonte: [2].

2º parte - filtragem mediana

Existem várias fontes de ruídos na imagem de entrada. Afim de eliminá-las, é aplicado um filtro passa média: *filter median,* para remover possíveis ruídos presente [1].

Essa técnica é aplicada com frequência em imagens e sinais com esse intuito, melhorando a imagem para detecção de bordas pois preserva as mesmas [5].

A ideia principal do filtro mediano, considerar cada *pixel* na imagem, por sua vez, verifica cada pixel de sua vizinhança para decidir se é ou não do seu arredores, em vez de simplesmente substituir o valor do *pixel* pela média dos valores de *pixels* vizinhos, ele substitui pela mediana desses valores. A mediana é calculada, ordenando primeiro todos os valores de *pixels* do bairro circundante em ordem numérica e, em seguida, substituindo o *pixel* sendo considerado com o valor do *pixel* médio. (Se a vizinhança em questão contiver um número par de *pixels*, a média dos dois valores do *pixel* médio é usada.) Assim ilustra a Figura 03, a imagem mostra o valor de *pixel* central de 150, que não é o bastante representativo dos pixels de sua vizinhança, e substitui pelo valor médio 124 [6].

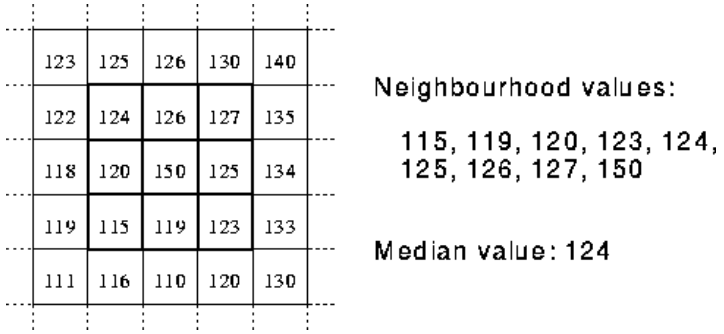

Figura 03: Cálculo filtro mediano.
Fonte: [6].

3º parte - detecção de bordas

Existem muitos detectores de bordas, visando o custo computacional e desempenho, os autores selecionaram o *zero-crossing detector: Difference of Exponential* (DOE), esse detector de bordas, quando o valor de sua escala é maior, apenas a borda da região relativamente maior é detectada [2]. Segundo (*Li, Bingcheng ; Zhao, Dongming,* 1995), o detector DOE é eficiente computacionalmente e seu desempenho de detecção de borda é maior do que o operador *Laplacian-of-Gaussian* (LOG) [7].

4º parte - ligação de bordas

Com base na função de energia H [1], é realizado o processo das ligações das bordas da imagem de entrada, como também, apaga as ligações cujo comprimento é menor que o especificado. Descompacta cada segmento de contorno da linha em um conjunto de segmentos com curvatura constante: linhas retas e arcos, armazenando em uma nova cadeia; para cada segmento, um outro contorno é pesquisado em uma região definida para que sua ligação tenha uma saliência global ideal. Assim as bordas são bem ligadas de modo que a informação do contorno da face é melhorada, como também outros ruído são diminuídos [1].

5º parte - correspondência de modelo

Com a observação dos contornos de uma cabeça humana, é notado que se aproxima da forma de uma elipse. Sabendo que o contorno da face não se assemelha a uma elipse perfeita, e como base na direção da borda, é proposto um modelo para detecção da face. Entretanto, essa elipse detectada na imagem não é perfeita, e deve contemplar as falhas na sua formação. Com base na detecção do contorno externo, que não é perfeito, é usando um anel elíptico para definir todos os pontos encontrados. Com base equação da elipse, é criado o modelo para comparação, que verifica se detecção ou da face humana [1].

2.1. Experimentos

O algoritmo proposto para testar o método descrito em [1], foi implementado na linguagem de programação *Python* (código disponível em anexos) e experimentado. Os autores apenas citam qual base de dados foi usada: (*MIT face database*, *CMU face database* e imagens da World Wide Web), e descrevem algumas características, mais adiante, porém, não disponibilizaram nenhum *link* do repositório das referidas bases.

Sendo assim, foi criada uma nova Base de Dados de Imagens (BDI) para realização dos testes, após não encontrar as referidas BDI em questão. A BDI usada no experimento são provenientes de: *Archive Computational Vision* [3] e Libor Spacek [4], selecionadas, pela sua distribuição livre para fins acadêmico e de fácil *download,* como também, contemplando imagens com fundo simples, moderado complexos, característica essenciais para se aproximar das imagens referidas pelos autores.

Características BDI *Archive Computational Vision*

Conjunto de dados do rosto frontal, colecionado por *Markus Weber* em *Instituto de Tecnologia da Califórniac,* diretório *Faces 1999 (Front)*: 450 imagens de rosto de pessoas do sexo masculino e feminino, com resolução de 896 x 592 *pixels,* em formato *.jpg, sendo, 27* ou mais pessoas únicas com diferentes iluminação, expressões e *background* complexo “fundo de escritório”.

Características BDI *Libor Spacek*

Conjunto de dados do rosto frontal, pessoas do sexo masculino e feminino, com e sem óculos, diretório *faces95 com* 3017 imagens, com *backgrounds* complexo, com resolução de 196 x 196 *pixels,* formato .jpg; no diretório faces96, com 1441 imagens, com *background* simples, com resolução de 180 x 200 *pixels no* formato .jpg.

Treinamento do algoritmo

Antes de submeter o algoritmo as referidas BDI’s, ele foi experimentado com 18 imagens da Web, com o propósito de estabelecer os melhores parâmetro possíveis, sendo 6 imagens com fundo simples, 6 com fundo moderado e 6 com fundo complexo.

Devido às bases de dados utilizada no experimento possuírem imagens em três escalas de cores RGB, foi necessário adicionar antes da melhoria de imagem (passo 01) um conversor de escala RGB para escala de cinza, entretanto os autores não especificam a escala de cores das imagens utilizadas, o que leva a deduzir que se tratava de escala tom de cinza, como exibido na Figura 06 e Figura 07, o processo de conversão de escala de cores.

Figura 06: Imagem de entrada.
Fonte: Própria

Figura 07: Imagem convertida para escala de cinza.
Fonte: Própria

1º parte - melhoria da imagem

Nesse primeiro momento, é verificado a imagem em escala de cinza o seu histograma antes da equalização, como mostra a imagem na Figura 08.

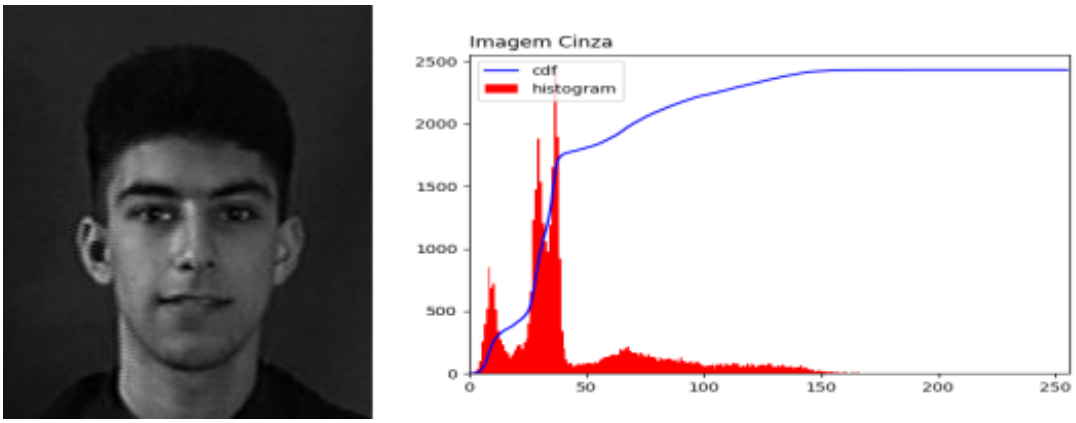

Figura 08: Gráfico da imagem, antes da equalização do histograma.
Fonte: Própria

No segundo momento, é aplicado a equalização do histograma na imagem em tom de cinza, como pode-se ser observada na Figura 09, o *background* simples da imagem, que era bastante escuro, é realçado, dando mais ênfase ao *foreground*, como também um clareamento maior na imagem.

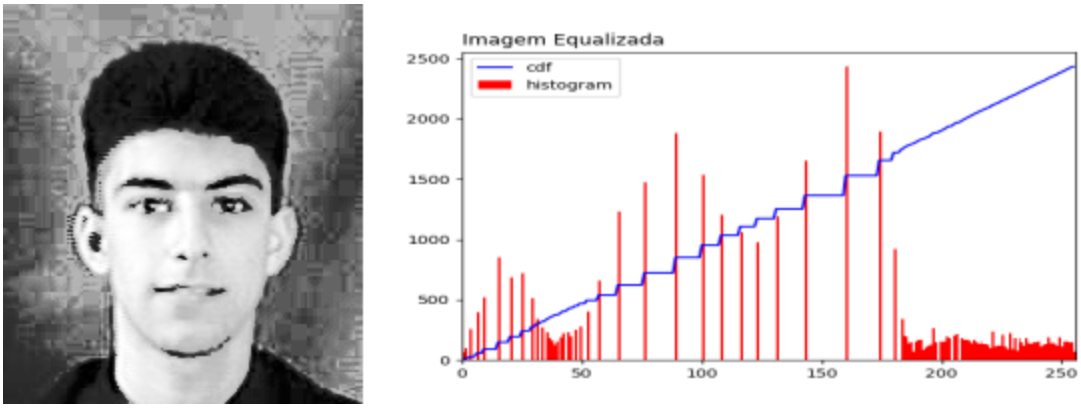

Figura 09: Gráfico da imagem, após a equalização do histograma.
Fonte: Própria

2º parte - filtragem mediana

Neste momento, é aplicando o filtro mediano, borrando um pouco a imagem, a fim de melhorar as imperfeições, eliminando ruídos. Esse efeito pode ser observado na Figura 10, assim, a imagem está pré-processada e pronta para ser usada na detecção de bordas.

Figura 10: Filtro median
Fonte: Própria

3º parte - detecção de bordas

Segundo os autores [1], existem diversos detectores de bordas. O detector de bordas DOE, foi escolhido pelos mesmo, visando custo e desempenho computacional, entretanto, o artigo tem enfoque na problemática de detecção facial, porém, não explana o suficiente para a implementação do mesmo, cita apenas, um livro, para maiores detalhes do funcionamento do detector de bordas em questão, tal livro não está disponível para minha consulta.

Devido a escassez de material referente a esse detector, o mesmo foi substituído pelo detector de borda *Canny,* para dar prosseguimento aos próximos passos. Na Figura

11, pode-se conferir o resultado em uma imagem binária da detecção de bordas pelo método *Canny*.

Figura 11: Detecção de bordas
Fonte: Própria

Esse detector de bordas processa a imagem visando detectar as mudanças repentina de brilho, tendo uma boa detecção, diminuindo a probabilidade de detectar bordas irreais e aumentado a probabilidade de detectar as bordas reais, ele usa um filtro aliciando para eliminar os ruídos, e encontra o gradiente de intensidade da imagem usando o procedimento análogo a *Sobel,* aplicando uma máscara de correlação nas direções X e Y, encontra a força e direção do gradiente e aplica uma supressão não máxima para remover os *pixels* que não são considerados, permanecendo apenas as linhas finitas (bordas candidatas), por fim, usando dois limiares, superior e inferior, estabelece parâmetros para aceitação e rejeição das bordas [8].

4º parte - ligação de bordas

Observe, na Figura 12, que na ligações de bordas, parte da bordas do que se parece uma orelha esquerda, é conectada, ou seja, houve uma melhoria, também alguns traços meio apagados são realçados, dando mais evidência e fortalecendo as bordas, entretanto, alguns ruídos são gerados nessa ligação, veja, do lado do que parece ser a orelha e o cabelo surgem uns traços proveniente da tentativa de ligar novos pontos de bordas.

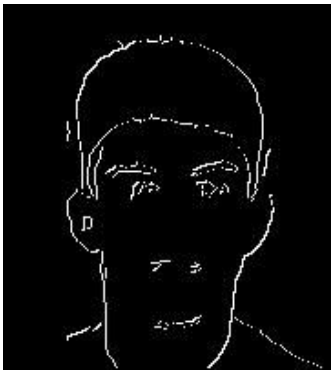

Figura 12: Ligação de bordas
Fonte: Própria

Para ligar essas bordas, é estabelecido dois limiares, uma baixo e outro alto, como também, cria duas imagens novas para armazenar os *pixels* de supressão não máxima, para conter todas as características para realização da normalização, localiza o próximo *pixel* não visitado e marca os *pixels* de bordas válidos, todos os que contêm características para serem ligados.

5º parte - correspondência de modelo

Com base na ligação e direção das bordas detectadas, por meio de um modelo elíptico é detectado a face na imagem, isso é feito por meio de um estabelecimento de um padrão elíptico, assim é analisado se as ligações das bordas no seu direcionamento formam uma elipse, caso essa região seja aprovada dentro desse modelo, a imagem é marcada uma elipse nesta região, caracterizando a detecção da face conforme ilustra a Figura 13:

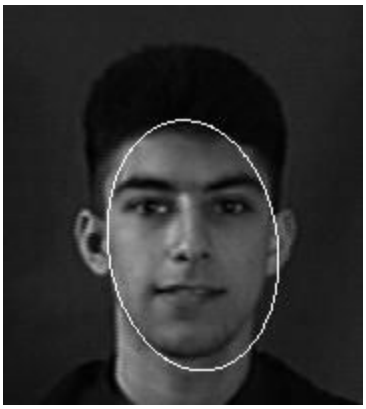

Figura 13: Detecção de face
Fonte: Própria

caso contrário, nada é feito na image, apenas contabilizado com um arquivo a qual não foi identificado nenhuma face.

Resultados de experimento e comparações

No primeiro caso de teste, o algoritmo foi submetido ao BDI *Archive Computational Vision* [3], com 432 arquivos, imagens com *backgroud* complexo e moderado, com escala de 896 x 592 *pixels*, com *backgroud* caracterizado de escritório na sua grande maioria, variando a orientação da cabeça, iluminação e zoom da câmera, conforme as características do teste feito pelos autores [1], diferenciando apenas os valores da escala de *pixels* da imagem.

Na imagem da Tabela 01, pode-se conferir resultados das comparações dos testes realizados. Onde:

Inexact[1]: significa que o anel elíptico passou por alguma parte do contorno e a face é incluída principalmente, mas a região não facial também foi incluída, ou algumas característica faciais foram perdidas.

Exact[1]*:* são as detecções corretas.

-	***Correct detection***		***False detection***	***No detection***
	Exact[1]	*Inexact*[1]		
Algoritmo de Wang J.; Tieniu T.	84.96%	50/432 11.57%	15/432 3.47%	0/432 0%
Minha implementação	34/432 7,87%	153/432 35,41%	176/432 40,74%	69/432 15,97%

Tabela 01: Comparação de resultados imagens *background* complexo
Fonte: Própria

No algoritmo de *Wang J.; Tieniu T.,* conforme a Tabela 0*1,* os resultados inexatos, segundos os autores, são causados na sua maioria pela complexidade do *background* da imagem que é a maior parte da base de dados, entretanto, o algoritmo mostra ter um bom desempenho na detecção, como 84,96% de exatidão.

Na minha implementação, o algoritmo teve uma alta taxa de detecções falsas, correspondendo a 40,74% de erros, pela análise das imagens, é observado que a maior parte dessas imagens o *background* é muito complexo de segmentar, devido o grande número de objetos no *background,* como: estantes de livros. Não foi possível detectar nenhum face em 69 imagens, um percentual de 15,97%, o que leva a crer, que se faz necessário realizar melhorias na segmentação. Apenas foi obtido 7,87% de exatidão nas detecções de face, um número não satisfatório, por outro lado, levar a apontar que o caminho está correto, pois houve um nível aceitável de detecções imprecisas, uma margem de 35,41% de faces detectadas com imprecisão. Na análise, foi possível observar que o ângulo de inclinação das cabeças influenciaram nas detecções imprecisas, como também o cabelo, acaba expandido a área de detecção.

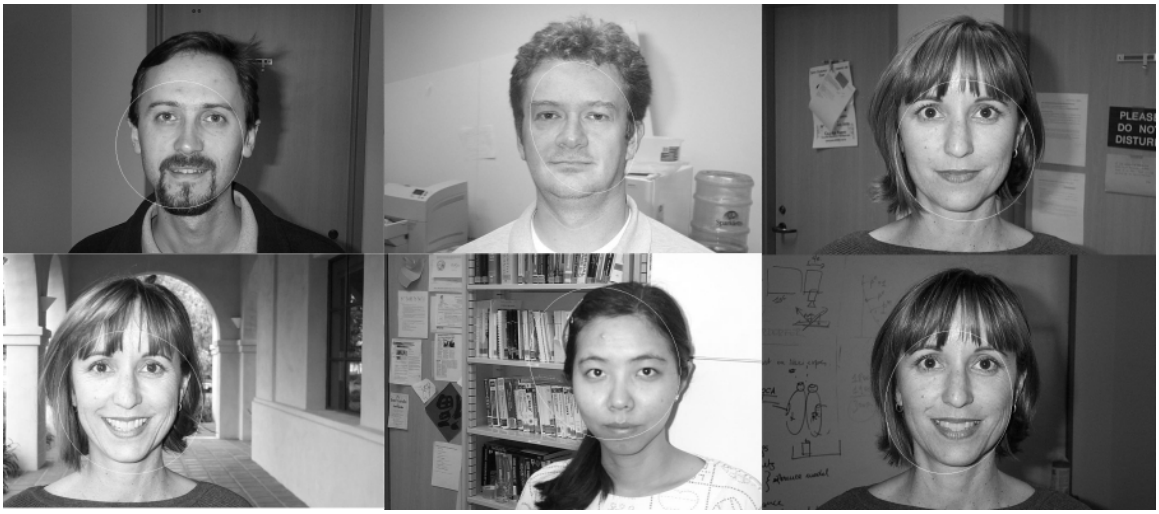

Figura 14: Resultados de detecção de imagens BDI *Archive Computational Vision*
Fonte: Própria

Um segundo caso de teste foi montado para averiguar com maior certeza, nesse sentido, foi elaborado um conjunto com 90 imagens, provinda do BDI *Libor Spacek* [4], do diretório *faces95,* foram selecionadas de forma aleatória 50 imagens com *background* simples formando o subconjunto¹, e no diretório *faces96,* foram selecionadas de forma aleatória 40 imagens com *background* complexo, formando o subconjuto², assim, obteve os seguintes resultados de desempenho, que pode-se ler na Tabela 02.

	Subconjunto¹ (bg simples)			Subconjunto² (bg complexo)		
	Exact	*Inexact*	*False*	*Exact*	*Inexact*	False
Algoritmo de Wang J.; Tieniu T.	42/50 84%	8/50 16%	0/50 0%	35/40 87,5%	0/40 0%	5/40 12,5%
Minha implementação	29/50 58%	15/50 30%	6/50 1,2%	6/40 15%	32/40 80%	2/40 5%

Tabela 02: Comparação de resultados imagens *background* simple e complexo
Fonte: Própria

No algoritmo de *Wang J. e Tieniu T.,* o teste empregado no subconjunto¹, seu algoritmo detectou todas as 50 imagens, porém 8 delas não teve exatidão devido ao tamanho das barbas e o grande ângulo de inclinação das cabeças. Desempenho muito bom, para imagens com backgroud simples.

No subconjunto² de imagens, pode detectar com exatidão 35 de 40 imagens equivalente a 87,5% de exatidão. Entretanto 5 anéis elípticos apresentaram detecção em locais errados, uma taxa de erro de 12,5%. O distúrbio de tantas bordas irrelevantes do fundo causa alta taxa de detecção falsa.

Meu algoritmo testado no subconjunto¹ *background* simple, obteve um desempenho razoavelmente bom, com quase 70% de detecções exatas, e uma taxa de 30% de detecção de faces, com uma precisão aceitável, tendo apenas 1,2% de detecções falsas, devido a ruídos presentes na imagem, que não foi eliminado por total, como também a presença de óculos, barbas na face, e bonés na cabeça, atrapalharam um pouco na detecção, que pode ser conferido na imagem da Figura 15.

Figura 15: Resultados de detecção de imagens subconjunto¹
Fonte: Própria

No subconjunto² com *background* complexo, o algoritmo novamente não teve um bom desempenho, entretanto, isso não significa que ele não consegue detectar as faces, como pode-se ser observado nos resultados na Tabela 02, ele conseguiu 80% de detecção de faces, porém com algumas imprecisões, como: pequenas partes da face não foram detectadas, houve um aumento da área da detecção devido a regiões de cabelos e objetos no fundo que não foram totalmente removidos. Apenas duas imagens obtiveram falsa detecção, uma taxa muito pequena de erro, comparado ao algoritmos dos autores que obtiveram 5 detecções falsas. Sendo assim, o algoritmo nesse segundo caso de teste, obteve um melhor resultado em relação ao primeiro caso de teste, porém, o meu algoritmo implementado não obteve um resultado igual ou melhor em todos os casos, em relação ao algoritmo dos autores [1].

Figura 16: Resultados de detecção de imagens subconjunto²
Fonte: Própria

2.2. Análise

Com o experimento demonstrado anteriormente, pode-se concluir o quanto é complicado elaborar o método proposto pelos autores *JianguoWang e TieniuTan*, como também, a grande eficácia que a sua técnica tem para detecção de face em imagens com background simples e complexo, parece robusto a variações de ruídos, entretanto, sofre falhas na variação do ângulo da cabeça, também, não apresentou como reagiria, caso as pessoas tivessem, chapéus ou bonés, alguns acessório na cabeça, pois ocorreu erros devido a barba presente na face.

3. Conclusões

Neste relatório, tratamos o problema de detecção de face com base em extração de característica de formas do contorno da face humana, apresentando o método proposto pelos autores [1], Esse método foi testado em duas bases de dados de imagens, com um total de 532 imagens com background simple e complexo, onde apresentou problemas de variação na detecção em face predominância de barbas muito grande, inclinação do ângulo da cabeça e *backgroud* muito complexo. Em contrapartida o método mostra ser bastante eficiente em imagens com *backgroud* simples.

Referências

[1] Jianguo Wang, Tieniu Tan, A new face detection method based on shape information, In Pattern Recognition Letters, Volume 21, Issues 6–7, 2000, Pages 463-471, ISSN 0167-8655.

[2] WIKIPEDIA. *Histogram equalization.* Disponível em: <https://en.wikipedia.org/wiki/Histogram_equalization>. Acesso 21/10/2017.

[3] ARCHIVE, *Computational Vision. Faces 1999 (Front)*. Disponível em: <http://www.vision.caltech.edu/html-files/archive.html>. Acesso 15/11/2017.

[4] SPACEK, Libor. *Description of the Collection of Facial Images.* Disponível em: <http://cswww.essex.ac.uk/mv/allfaces/index.html>. Acesso 13/11/2017.

[5] *WIKIPEDIA*. *Median filter.* Disponível em: <https://en.wikipedia.org/wiki/Median_filter>. Acesso 13/11/2017.

[6] HIPR2. Median Filter. <https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm>. 19/11/2017.

[7] *LI, Bingcheng; ZHAO, Dongming. Difference-of-exponential detector for extracting edges.* Disponível em: <http://adsabs.harvard.edu/abs/1995SPIE.2501..912L>. Acesso 14/10/2017.

[8] DOC, OpenCV. Canny Edge Detector. Disponível em: <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html>. 19/11/2017.

Anexos

Código fonte implementado.

```
# coding: utf-8
# !/usr/bin/python
# Python 2/3 compatibility

'''
Link Repositório Github: https://github.com/rodriguesfas/visioncomputer
'''

# Import lib's.
import cv2
import numpy as np
import glob
import scipy.ndimage as ndi
import scipy
import Image
import math
from math import pi
import time
from random import choice
from matplotlib import pyplot as plt

'''
    Variáveis globais
'''
START_TIME = time.time()
SIGMA = 2,2
PI = 1.14

'''
    1 ativar
    0 desativar
    Obs.: usar apenas com diretório unitário.
'''
TEST_UNIT = 0

'''
    1 ativar
    0 desativar
    Obs.: para exibir gráfico.
'''
GRAPHIC = 0
```

```
df_yes = 0     # faces detectadas.
df_no = 0      # possíveis faces.
all_df = 0     # total de faces detectadas.
all_no_df = 0  # total de possíveis faces.
allfiles = 0   # total de arquivos analisados.



'''
    Configuração do caminho do diretório de imagens.
'''

# ATENÇÃO, esse diretório é destinado apenas para imagens de treinamento do
algoritimo.
#dir_files = '../src/img/input/database/training/unitario'

# Diretórios das bases de dados para processamento.
#dir_files = '../src/img/input/database/test_01/bg_complexo'

#dir_files = '../src/img/input/database/test_02/bg_simples'
dir_files = '../src/img/input/database/test_02/bg_complexo'

img_file = [i for i in glob.glob(dir_files+"/*.jpg")]
img_file.sort()
images = []

'''
    Gerador de código de identificação, para salvar as imagens processadas.
'''
def randomIDImg(size):
    data = '0123456789'
    code = ''

    for char in xrange(size):
        code += choice(data)

    return  code

'''
    Exibe imagens processadas na tela.
    Usar, somente em modo de teste, quando estiver exibindo as imagens na janela.
    Recomenda-se testar com 1 imagens apenas, para visualizar todas as janelas que
serão abertas passo a passo do processo.
'''
def showImg(title, img):
    if(TEST_UNIT):
        cv2.imshow(title, img)

# outHist - Exibe resultado do histograma da imagem em questão.
```

```
def outHist(title, img):
    hist, bins = np.histogram(img.flatten(), 256, [0, 256])

    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max()/ cdf.max()

    plt.plot(cdf_normalized, color='b')
    plt.hist(img.flatten(), 256, [0, 256], color='r')
    plt.xlim([0, 256])
    plt.title(title, loc='left')
    plt.legend(('cdf', 'histogram'), loc='upper left')
    plt.show()

'''
    Loop
'''
print('Iniciado análise de imagens ;)')

for file in img_file:

    id_img = randomIDImg(10)

    f = cv2.imread(file)
    images.append(f)
    print('arquivo processado: '+file)


    img_orig = cv2.imread(file)
    showImg('Image Original', img_orig)

    # Verifica se a imagem é RGB antes de converter para escala de cinza.
    if img_orig.shape[-1] == 3:                    # color image
        b, g, r = cv2.split(img_orig)              # get b, g, r
        img_orig = cv2.merge([r, g, b])            # switch it to rgb
        img_gray = cv2.cvtColor(img_orig, cv2.COLOR_BGR2GRAY)

        cv2.imwrite('../src/img/output/processing/'+id_img+'_img_gray.jpg', img_gray)
        showImg('Step 00 - Image Cinza', img_gray)
    else:
        img_gray = img_orig



    '''
    Step 01 - Image Enhancement

    As imagens de entrada, podem ter um contraste muito fraco devido à limitação das
condições
    de iluminação. Nesse passo, a equalização do histograma é usada para melhorar o
contraste
```

```
    da imagem original.
    '''
    img_hist = cv2.equalizeHist(img_gray)
    cv2.imwrite('../src/img/output/processing/'+id_img+'_img_hist.jpg', img_hist)
    showImg('Step 01 - Image Enhancement (Equalize Histogram)', img_hist)

    if(GRAPHIC):
        outHist("Imagem Cinza", img_gray)
        outHist("Imagem Equalizada", img_hist)


    '''
    Step 02 - Median Filtering (LPF)

    Devido ao passo anterior, é óbvio que o número de pontos de contorno do rosto
podem aumentar,
    o que significa, que a informação facial foi fortalecida. O ruído também foi
aprimorado.
    Por meio da filtragem, podem ser apagados as fontes de ruídos presente na imagem,
aplicando um
    median filtering.
    '''
    img_hist = cv2.imread('../src/img/output/processing/'+id_img+'_img_hist.jpg', 0)
    img_blur = cv2.medianBlur(img_hist, 5)

    cv2.imwrite('../src/img/output/processing/'+id_img+'_img_blur.jpg', img_blur)
    showImg('Step 02 - Median Filtering', img_blur)

    '''
    Step 03 - Edge Detection

    Existem muitos detectores de bordas, considerando não considerando o custo
computacional e o desempenho foi usado
    o zero-crossing detector: Canny Edge Detection (DoG).
    '''

    img_blur = '../src/img/output/processing/'+id_img+'_img_blur.jpg'
    img_blur = Image.open(img_blur)
    img_data = np.array(img_blur, dtype = float)

    img_median = ndi.filters.median_filter(img_data, SIGMA)

    # imagem vazia
    sobelout = Image.new('L', img_blur.size)
    gradx = np.array(sobelout, dtype = float)
    grady = np.array(sobelout, dtype = float)

    sobel_x = [
        [-1, 0, 1],
        [-2, 0, 2],
```

```
        [-1, 0, 1]
        ]

    sobel_y = [
        [-1, -2, -1],
        [ 0,  0,  0],
        [ 1,  2,  1]
        ]

    width = img_blur.size[1]
    height = img_blur.size[0]

    #calculate |img_median| and dir(img_median)

    for x in range(1, width-1):
        for y in range(1, height-1):
            px = (sobel_x[0][0] * img_median[x-1][y-1]) + (sobel_x[0][1] *
img_median[x][y-1]) + \
                 (sobel_x[0][2] * img_median[x+1][y-1]) + (sobel_x[1][0] *
img_median[x-1][y]) + \
                 (sobel_x[1][1] * img_median[x][y]) + (sobel_x[1][2] *
img_median[x+1][y]) + \
                 (sobel_x[2][0] * img_median[x-1][y+1]) + (sobel_x[2][1] *
img_median[x][y+1]) + \
                 (sobel_x[2][2] * img_median[x+1][y+1])

            py = (sobel_y[0][0] * img_median[x-1][y-1]) + (sobel_y[0][1] *
img_median[x][y-1]) + \
                 (sobel_y[0][2] * img_median[x+1][y-1]) + (sobel_y[1][0] *
img_median[x-1][y]) + \
                 (sobel_y[1][1] * img_median[x][y]) + (sobel_y[1][2] *
img_median[x+1][y]) + \
                 (sobel_y[2][0] * img_median[x-1][y+1]) + (sobel_y[2][1] *
img_median[x][y+1]) + \
                 (sobel_y[2][2] * img_median[x+1][y+1])
            gradx[x][y] = px
            grady[x][y] = py

    sobeloutmag = scipy.hypot(gradx, grady)
    sobeloutdir = scipy.arctan2(grady, gradx)

    scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewmag.jpg',
sobeloutmag)
    scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewdir.jpg',
sobeloutdir)

    for x in range(width):
        for y in range(height):
            if (sobeloutdir[x][y]<22.5 and sobeloutdir[x][y]>=0) or \
               (sobeloutdir[x][y]>=157.5 and sobeloutdir[x][y]<202.5) or \
```

```
               (sobeloutdir[x][y]>=337.5 and sobeloutdir[x][y]<=360):
                sobeloutdir[x][y]=0
            elif (sobeloutdir[x][y]>=22.5 and sobeloutdir[x][y]<67.5) or \
                 (sobeloutdir[x][y]>=202.5 and sobeloutdir[x][y]<247.5):
                sobeloutdir[x][y]=45
            elif (sobeloutdir[x][y]>=67.5 and sobeloutdir[x][y]<112.5)or \
                 (sobeloutdir[x][y]>=247.5 and sobeloutdir[x][y]<292.5):
                sobeloutdir[x][y]=90
            else:
                sobeloutdir[x][y]=135


scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewdirquantize.jpg', sobeloutdir)

    '''
    Step 04 - Edge Linking

    Conecta as bordas pequenas da imagem com as bordas grandes, com base na direção das bordas, por meio do gradiente de bordas com
    o operador Sobel.
    '''
    mag_sup = sobeloutmag.copy()

    for x in range(1, width-1):
        for y in range(1, height-1):
            if sobeloutdir[x][y]==0:
                if (sobeloutmag[x][y]<=sobeloutmag[x][y+1]) or \
                   (sobeloutmag[x][y]<=sobeloutmag[x][y-1]):
                    mag_sup[x][y]=0
            elif sobeloutdir[x][y]==45:
                if (sobeloutmag[x][y]<=sobeloutmag[x-1][y+1]) or \
                   (sobeloutmag[x][y]<=sobeloutmag[x+1][y-1]):
                    mag_sup[x][y]=0
            elif sobeloutdir[x][y]==90:
                if (sobeloutmag[x][y]<=sobeloutmag[x+1][y]) or \
                   (sobeloutmag[x][y]<=sobeloutmag[x-1][y]):
                    mag_sup[x][y]=0
            else:
                if (sobeloutmag[x][y]<=sobeloutmag[x+1][y+1]) or \
                   (sobeloutmag[x][y]<=sobeloutmag[x-1][y-1]):
                    mag_sup[x][y]=0


scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewmagsup.jpg', mag_sup)

    m = np.max(mag_sup)
    th = 0.2*m
    tl = 0.1*m
```

```
    gnh = np.zeros((width, height))
    gnl = np.zeros((width, height))

    for x in range(width):
        for y in range(height):
            if mag_sup[x][y]>=th:
                gnh[x][y]=mag_sup[x][y]
            if mag_sup[x][y]>=tl:
                gnl[x][y]=mag_sup[x][y]

    scipy.misc.imsave('../src/img/output/processing/'+id_img+'.jpg', gnl)

    gnl = gnl-gnh


scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewgnlafterminus.
jpg', gnl)
    scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewgnh.jpg',
gnh)


    #
    def traverse(i, j):
        x = [-1, 0, 1, -1, 1, -1, 0, 1]
        y = [-1, -1, -1, 0, 0, 1, 1, 1]

        for k in range(8):
            if gnh[i+x[k]][j+y[k]]==0 and gnl[i+x[k]][j+y[k]]!=0:
                gnh[i+x[k]][j+y[k]]=1
                traverse(i+x[k], j+y[k])



    for i in range(1, width-1):
        for j in range(1, height-1):
            if gnh[i][j]:
                gnh[i][j]=1
                traverse(i, j)

    scipy.misc.imsave('../src/img/output/processing/'+id_img+'_img_cannynewout.jpg',
gnh)

    showImg('Step 03/04 - Edge Detection/Edge Linking', gnh)



    '''
    Step 05 - Template Matching
```

```
    '''
    edge_image =
cv2.imread('../src/img/output/processing/'+id_img+'_img_cannynewout.jpg', 0)

    img_dilate = cv2.dilate(edge_image, np.ones((3, 3)), iterations=1)
    #img_eroded = cv2.erode(img_dilate, np.ones((3, 3)), iterations=3)

    showImg("Dilate", img_dilate)

    labels, count = ndi.label(img_dilate)

    for lab, idx in enumerate(ndi.find_objects(labels.astype(int)), 1):

        sy = idx[0].start
        sx = idx[1].start

        y, x = np.where(labels[idx] == lab)

        # elipse externa
        ellipse_outer = cv2.fitEllipse(np.column_stack((x+sx, y+sy)))

        '''
            @ Parâmetros de uma elipse | elliptical ring

                xc: coordenada x do centro
                yc: coordenada y do centro
                a: semi-eixo principal
                b: semi-eixo secundário
                theta: ângulo de rotação
        '''
        # anel eliptico
        (xc, yc), (a, b), theta = cv2.fitEllipse(np.column_stack((x+sx, y+sy)))

        # calcula os paramentro do pontos que formam a elipse.
        isElipse = ( (xc**2) / (b**2) ) + ( (yc**2) / (a**2) )

        # Calcula ŕea da elipse.
        area = a * b * PI

        # Mostra características da elipse.
        if(TEST_UNIT):
            print("xc: ", xc)
            print("yc: ", yc)
            print("a: ", a)
            print("b: ", b)
            print("theta: ", theta)
            print("value elipse: ", isElipse)
            print("area: ", area)
            print("all data ellp outer: ", ellipse_outer)
```

```
        # Verificar se é uma elipse e estabelece um tamanho da área para evitar err.
        if(isElipse <= 2): #isElipse <= 2  and area < 36255

            if(TEST_UNIT):
                # elipse externa | imagem cinza.
                img_gray = cv2.ellipse(img_gray, ellipse_outer, (255, 255, 255), 1)

                # imagem binária com detecção facial | elipse externa.
                img_dilate = cv2.ellipse(img_dilate, ellipse_outer, (255, 255, 255),
1)

            # imagem cinza com detecção facial
            img_gray = cv2.ellipse(img_gray, ((xc, yc), (a/1.3, b/1.6), theta), (255,
255, 255), 1) #elipse interna
            #img_gray = cv2.ellipse(img_gray, ellipse_outer, (255, 255, 255), 1)
#elipse externa

            if(TEST_UNIT):
                img_dilate = cv2.ellipse(img_dilate, ((xc, yc), (a/1.3, b/1.6),
theta), (255, 255, 255), 1)

            df_yes+=1

        else:
            df_no+=1

    print "Encontrei {0} face(s) nesta imagem.".format(df_yes)

    if(df_no != 0):
        print "Encontrei {0} possíveis face(s) nesta imagem.".format(df_no)

    scipy.misc.imsave('../src/img/output/result/'+id_img+'_img_detection.jpg',
img_gray)
    showImg("Step 05 - Template Matching", img_gray)

    if(TEST_UNIT): #TEST_UNIT
        scipy.misc.imsave('../src/img/output/result/'+id_img+'_img_detection2.jpg',
img_dilate)
        showImg("Detec face img BIN", img_dilate)


    # acumulador
    all_df = all_df + df_yes
    all_no_df = all_no_df + df_no

    # reseta variáveis
    df_yes = 0
    df_no = 0

    # contador de arquivos.
```

```
    allfiles+=1

END_TIME = time.time()
TIME_TAKEN = END_TIME - START_TIME

print('Fim de análise :)')
print('-------------------------------------------------------')
print 'Tempo total de excução: ', TIME_TAKEN
print "{0} arquivo(s) analisado(s) no total.".format(allfiles)
print "Encontrei {0} face(s) no total :)".format(all_df)
print "Encontrei {0} possíveis face(s) no total ;)".format(all_no_df)

'''
    Exit
    Finaliza o processo ao clicar em 0.
'''
if(TEST_UNIT):
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```