UNIVERVISADE ESTADUAL DE CAMPINAS - UNICAMP
INSTITUTO DE COMPUTAÇÃO - IC

Introdução ao Processamento de Imagem Digital

Prof. Hélio Pedrini

Trabalho 5

Nicolas Guilhermo Silva Moliterno

232389

Campinas - SP

2022

Lista de Figuras

Sumário

1 Introdução

O objetivo deste trabalho é detectar movimentos por meio da comparação de imagens capturadas por uma câmera de vídeo. A identificação de movimentos na cena é realizada pela verificação da diferença entre o quadro atual e o quadro anterior. Caso os conteúdos dos quadros sejam iguais, então não há mudanças aparentes na cena. Quando há diferenças, os locais de mudanças (movimento) serão apresentados.

2 Materiais e Metodologia

Foi utilizado os pacotes python OpenCv e matplotlib, para o auxílio de cálculos e métodos para analisar as imagens.

Funções utilizadas:

cv2.VideoCapture(...) . Use cv2.VideoCapture() para obter um objeto de captura de vídeo para a câmera.

cv2.cvtColor(...) : O método cv2.cvtColor() é usado para converter uma imagem de um espaço de cor para outro. Existem mais de 150 métodos de conversão de espaço de cores disponíveis no OpenCV.

cv2.GaussianBlur(...) : Técnicas de suavização de imagem ajudam a reduzir o ruído. No OpenCV, a suavização da imagem (também chamada de desfoque) pode ser feita de várias maneiras. Neste trabalho utilizamos o filtro gaussiano para suavização de imagem. Os filtros gaussianos têm as propriedades de não ultrapassar a entrada de uma função degrau, enquanto minimizam o tempo de subida e descida. Em termos de processamento de imagem, quaisquer bordas nítidas nas imagens são suavizadas, minimizando muito o desfoque.

cv2.threshold(...) : Thresholding é a binarização de uma imagem. Em geral, procuramos converter uma imagem em tons de cinza em uma imagem binária, onde os píxeis são 0 ou 255. Um exemplo de limite simples seria selecionar um valor limite T e, em seguida, definir todas as intensidades de píxel menores que T como 0 e todos os valores de píxel maiores que T como 255. Dessa forma, podemos criar uma representação binária da imagem.

cv2.dilate(...) : As transformações morfológicas são algumas operações simples baseadas na forma da imagem. Normalmente é executado em imagens binárias. Ele precisa de duas entradas, uma é a nossa imagem original, a segunda é chamada de elemento estruturante ou núcleo que decide a operação. Utilizando a dilatação, um elemento de píxel é '1' se pelo menos um píxel sob o kernel for '1'. Assim, aumenta a região branca na imagem ou aumenta o tamanho do objeto em primeiro plano. Normalmente, em casos como remoção de ruído, a erosão é seguida de dilatação. Porque a erosão remove os ruídos brancos,

mas também encolhe nosso objeto. Então nós dilatamos. Como o ruído acabou, eles não voltarão, mas nossa área de objeto aumenta. Também é útil para unir partes quebradas de um objeto.

cv2.findContours(...) : Basicamente, este método descobre todos os pontos de limite da forma na imagem.

cv2.boundingRect(...) : É uma função usada para criar um retângulo aproximado com a imagem. O uso principal desta função é destacar a área de interesse após obter a forma externa da imagem. Com marcações adequadas, os usuários podem destacar facilmente o aspecto desejado em uma imagem.

cv2.contourArea(...) : Retorna a área dentro de um contorno.

3 Resultados

3.1 Captura do Video

Primeiro utilizamos a função de captura para gravar um pequeno video utilizando a webcam do computador.

```python
import sys
import signal
import cv2

end = False

# ------------------------------------
def signal_handler(signal, frame):
    global end
    end = True

# ------------------------------------
cap = cv2.VideoCapture(0)

# Tenta abrir a webcam, e já falha se não conseguir
if not cap.isOpened():
    print('Não foi possível abrir a web cam.')
    sys.exit(-1)

# Cria o arquivo de video de saída
fourcc = cv2.VideoWriter_fourcc(*'MP4V')
out = cv2.VideoWriter('./video/teste.mp4',fourcc, 20.0, (640,480))

# Captura o sinal de CTRL+C no terminal
signal.signal(signal.SIGINT, signal_handler)
print('Capturando o vídeo da webcam -- pressione Ctrl+C para encerrar...')

# Processa enquanto o usuário não encerrar (com CTRL+C)
while(not end):
    ret, frame = cap.read()
    if ret:
        out.write(frame)
    else:
        print('Oops! A captura falhou.')
        break

print('Captura encerrada.')

# Encerra tudo
cap.release()
out.release()
```

Figura 1: Código captura de video.

3.2 Conversão do Video em Escala de Cinza

Após isso convertemos o video para escala de cinza para realizar as operações.

```python
# importing the module
import cv2
import numpy as np

# reading the vedio
source = cv2.VideoCapture('./video/output.mp4')

# We need to set resolutions.
# so, convert them from float to integer.
frame_width = int(source.get(3))
frame_height = int(source.get(4))

size = (frame_width, frame_height)

result = cv2.VideoWriter('./video/gray.avi',
            cv2.VideoWriter_fourcc(*'MJPG'),
            10, size, 0)

# running the loop
while True:

    # extracting the frames
    ret, img = source.read()

    # converting to gray-scale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # write to gray-scale
    result.write(gray)

    # displaying the video
    cv2.imshow("Live", gray)

    # exiting the loop
    key = cv2.waitKey(1)
    if key == ord("q"):
        break

# closing the window
cv2.destroyAllWindows()
source.release()
```

Figura 2: Código conversão do video em escala de cinza.

3.3 Convertendo o Video em Frames de Imagens

Para realizar as operações de OpenCV no video, convertemos em frames.

```python
import cv2
import sys

vidcap = cv2.VideoCapture('./video/gray.avi')
success,image = vidcap.read()
count = 0
while success:
  cv2.imwrite("./images/frame%d.jpg" % count, image)     # save frame as JPEG file
  success,image = vidcap.read()
  print('Read a new frame: ', success)
  count += 1
```

Figura 3: Conversão em frames.

Figura 4: Resultado em cinza.

3.4 Filtro Gaussiano

Aplicando o filtro gaussiano para suavizar ruídos nas imagens.

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt


for image in image_list_gau:
    count = (image.split('frame')[1]).split('.')[0]

    img = cv2.imread(image)

    blur = cv2.GaussianBlur(img,(5,5),0)

    cv2.imwrite(f'./suavizacao/frame_blur{count}.png', blur)

    plt.subplot(121),plt.imshow(img),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(blur),plt.title('Suavizada')
    plt.xticks([]), plt.yticks([])

plt.show()


```

Figura 5: Código filtro gaussiano.

3.5 Limiarizações

Com a imagem suavizada, realizamos limiarizações. Limiarizações diferentes são mostradas com o mesmo intervalo adequado para os limiares, mas para prosseguir com os resultados, apenas a limiarização binaria é utilizada. Como foram realizadas diferentes limiarizações, o tempo de execução do algoritmo foi longo.

Figura 6: Imagem suavizada.

```python
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

for image in image_list_limi:
    count = (image.split('frame')[1]).split('.')[0]

    img = cv.imread(image, 0)
    ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
    ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
    ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
    ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
    ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
    titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
    images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
    for i in range(6):
        plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
        plt.title(titles[i])
        plt.xticks([]),plt.yticks([])
        plt.savefig(f'./limiarizacao/frame_bin{count}.png')
```

Figura 7: Código para realizar as limiarizações.

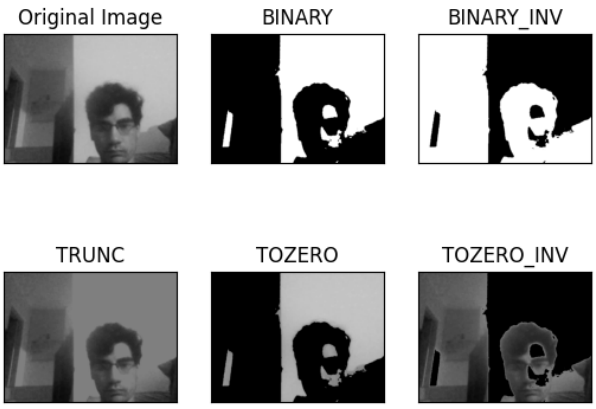

Figura 8: Imagens limiarizadas.

3.6 Dilatação

Com os resultados da limiarização binaria é realizada uma dilatação de forma que pixels são adicionados as bordas.

```python
import cv2 as cv, cv2
import numpy as np
from matplotlib import pyplot as plt

for image in image_list_dila:
    count = (image.split('frame')[1]).split('.')[0]

    img = cv.imread(image, 0)
    kernel = np.ones((5, 5), 'uint8')
    dila = cv2.dilate(img, kernel, iterations=1)
    cv2.imwrite(f'./dilatacao/frame{count}.png', dila)

    plt.subplot(121),plt.imshow(img,'gray',vmin=0,vmax=255),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(dila,'gray',vmin=0,vmax=255),plt.title('Dilatação')
    plt.xticks([]), plt.yticks([])
```

Figura 9: Código para a dilatação.

Figura 10: Resultado da dilatação.

3.7 Contornos

Com a imagem dilatada, conseguimos encontrar as regiões de contorno da imagem.

3.8 Retângulos

Para encontrar os retângulos envolventes nas imagens do video, não foi possível salvar os resultados, apenas visualizar em tela. Mas foi realizado em uma imagem exemplo para mostrar os retângulos envolventes dos contornos.

```python
import cv2 as cv
import cv2
import numpy as np
from matplotlib import pyplot as plt

for image in image_list_con:
    count = (image.split('frame')[1]).split('.')[0]

    img = cv.imread(image, 0)

    edged = cv2.Canny(img, 30, 200)

    contours, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    cv2.imwrite(f'./contornos/frame{count}.png', edged)

    cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
    #cv2.imshow('Conto', img)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    plt.subplot(121),plt.imshow(edged,'gray',vmin=0,vmax=255),plt.title('Contornos')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img,'gray',vmin=0,vmax=255),plt.title('Original')
    plt.xticks([]), plt.yticks([])
```

Figura 11: Código para a região de contornos.

Figura 12: Região de contornos da imagem.

3.9 Área de contorno

O mesmo método da região de retângulos, foi utilizado para encontrar as áreas de contorno. A imagem de frames do video é visualizada em tela, mas para exemplificar os resultados foi utilizada uma imagem de exemplo.

3.10 Detecção de Movimento

Para um adicional aos resultados, como não foi possível salvar as regiões de contorno sobrepostas as imagens. É realizada uma detecção de faces ao video gravado. É uma característica comum que entre todas as faces à região dos olhos é mais escura do que a região das bochechas. Portanto, uma característica comum para a detecção de face é um conjunto de dois retângulos adjacentes que ficam na região dos olhos e acima da região das bochechas. A posição desses retângulos é definida em relação a uma janela de detecção que age como uma caixa delimitadora

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

filepath = "./retangulos/australia.png"
australia = cv2.imread(filepath)
australia_grey = cv2.cvtColor(australia, cv2.COLOR_BGR2GRAY)
australia_thresh = cv2.threshold(australia_grey,226,255,cv2.THRESH_BINARY)[1]
australia_binary = cv2.bitwise_not(australia_thresh)


x1,y1,w,h = cv2.boundingRect(australia_binary)
x2 = x1+w
y2 = y1+h


start = (x1, y1)
end = (x2, y2)
colour = (255, 0, 0)
thickness = 1
rectangle_img = cv2.rectangle(australia, start, end, colour, thickness)
print("x1:", x1, "x2:", x2, "y1:", y1, "y2:", y2)
plt.imshow(rectangle_img, cmap="gray")
cv2.imwrite("./retangulos/retangulos.png", rectangle_img)
```

Figura 13: Código para retângulo nos contornos.

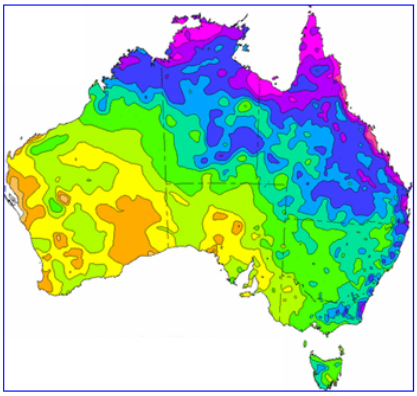

Figura 14: Resultado dos retângulos de contorno.

```python
import cv2 as cv
import cv2
import numpy as np
from matplotlib import pyplot as plt

for image in image_list_con:
    count = (image.split('frame')[1]).split('.')[0]

    img = cv.imread(image, 0)

    edged = cv2.Canny(img, 30, 200)

    contours, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    cv2.imwrite(f'./contornos/frame{count}.png', edged)

    cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
    #cv2.imshow('Conto', img)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    plt.subplot(121),plt.imshow(edged,'gray',vmin=0,vmax=255),plt.title('Contornos')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img,'gray',vmin=0,vmax=255),plt.title('Original')
    plt.xticks([]), plt.yticks([])
```

Figura 15: Código para área de contorno.

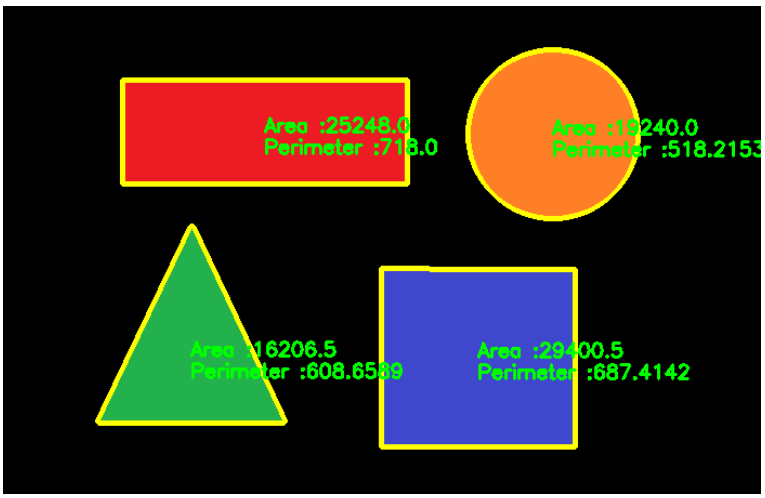

Figura 16: Áreas de contorno das imagens.

para o objeto alvo (a face, neste caso). O código abaixo realiza essa operação e exibe o video em tela. Para mostrar os resultados, uma captura de tela dos vídeos foi realizada.

```python
import cv2

def redim(img, largura): #função para redimensionar uma imagem
    alt = int(img.shape[0]/img.shape[1]*largura)
    img = cv2.resize(img, (largura, alt), interpolation =
    cv2.INTER_AREA)
    return img


#Cria o detector de faces baseado no XML
df = cv2.CascadeClassifier('./video/haarcascade_frontalface_default.xml')
#Abre um video gravado em disco
camera = cv2.VideoCapture('./video/output.mp4')

while True:
    #read() retorna 1-Se houve sucesso e 2-O próprio frame
    (sucesso, frame) = camera.read()
    if not sucesso: #final do video
        break
    #reduz tamanho do frame para acelerar processamento
    frame = redim(frame, 320)
    #converte para tons de cinza
    frame_pb = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #detecta as faces no frame
    faces = df.detectMultiScale(frame_pb, scaleFactor = 1.1, minNeighbors=3, minSize=(20,20), flags=cv2.CASCADE_SCALE_IMAGE)
    frame_temp = frame.copy()
    for (x, y, lar, alt) in faces:
        cv2.rectangle(frame_temp, (x, y), (x + lar, y + alt), (0, 255, 255), 2)
    #Exibe um frame redimensionado (com perca de qualidade)
    cv2.imshow("Encontrando faces...", redim(frame_temp, 640))
    #Espera que a tecla 's' seja pressionada para sair
    if cv2.waitKey(1) & 0xFF == ord("s"):
        break
#fecha streaming
camera.release()
cv2.destroyAllWindows()
```

Figura 17: Código para a captura de faces.

4 Conclusão

Este trabalho teve por objetivo propor um método de detecção de movimentos em imagens, a partir dos contornos que a imagem possui, através do uso de algoritmos de aprendizagem de máquina, utilizando as principais características usadas nesse tipo de aplicação extraídas de imagens 2D. O que pode ser concluído é que a visão computacional aliada aos algoritmos de processamento digital de imagem é altamente viável, não se limitando à tecnologia, e sim a hardwares capazes de suportar as necessidades do sistema proposto. Como visto no método para as diferentes limiarizações.

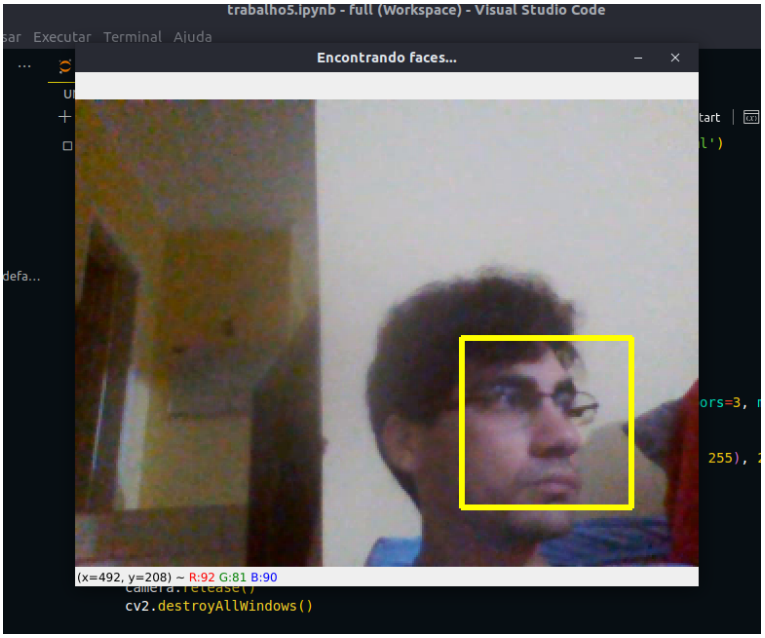

Figura 18: Primeira captura de imagem do video exibido com a captura de imagens.

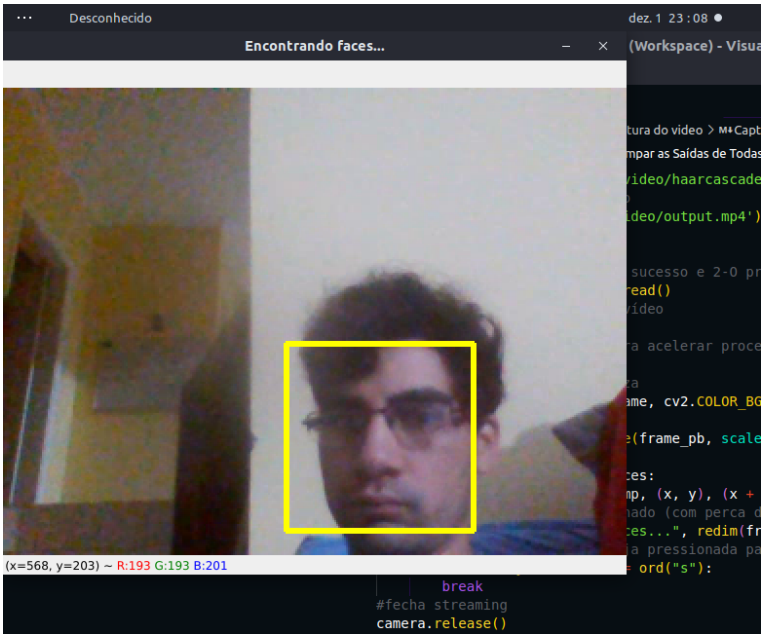

Figura 19: Segunda captura de imagem do video exibido com a captura de imagens.

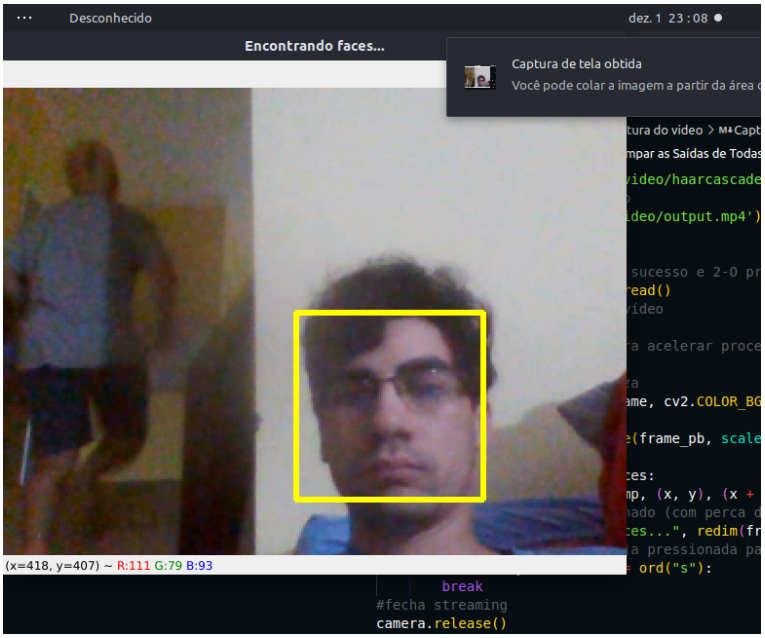

Figura 20: Terceira captura de imagem do video exibido com a captura de imagens.