THALLES SILVA DE FREITAS - 2016009978

Processamento digital de imagens

Visão computacional, conceito e aplicações

Itabira
Março, 2021

THALLES SILVA DE FREITAS - 2016009978

Processamento digital de imagens

Visão computacional, conceito e aplicações

Relatório referente ao trabalho final da disciplina de Computação gráfica e processamento digital de imagens, da Universidade Federal de Itajubá - Campus Itabira, para a obtenção de créditos.
Prof. Msc. Natasha Sayuri Dias Nakashima

Universidade Federal de Itajubá – Unifei
Graduação em Engenharia de Computação

Itabira
Março, 2021

1. Introdução

O conceito de visão computacional resumidamente pode ser descrito como a maneira que computadores interpretam imagens e vídeos que são passados a códigos programados para fazer o seu processamento, estes códigos podem ser escritos em diversas linguagens, a técnica também pode ser considerada um tipo de inteligência artificial, visto que simula a visão humana e a capacidade que o cérebro humano tem de diferenciar objetos, cores, fazer contagens entre outras funções. Segundo PEREIRA et al. (2018) [1], são utilizados diversos conhecimentos de outras áreas, como processamento de imagens, aprendizado de máquina, reconhecimento de padrões, e até outras como biologia e psicologia, para termos formado o conceito de visão computacional.

A ideia central desse trabalho será mostrar algumas ferramentas, aplicações e desenvolver de forma prática e simplificada um trabalho relacionado a computação visual, este trabalho utilizará técnicas de processamento de imagens em um vídeo, com o objetivo de identificar objetos (no caso pessoas) que passam em um determinado espaço.

2. Ferramentas

Nesse meio com o desenvolvimento da tecnologia, bibliotecas completas foram sendo implementadas, como OpenCV, SimpleCV, Libccv, ROS, FastCV e BoofCV.

A mais popular é a OpenCV (Open Source Computer Vision Library) que é uma biblioteca de código aberto para visão computacional. De acordo com o site [2], a biblioteca OpenCV foi construída para criar uma infraestrutura comum para aplicações que usam visão computacional e acelerar o uso da percepção das máquinas em produtos comerciais. Possuindo uma licença de código aberto, OpenCV facilita o uso das empresas que podem modificar o código livremente. Alguns exemplos de empresas que utilizam a biblioteca são: Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota além de Startups e empresas de menor porte, com a utilização as companhias ainda ajudam no desenvolvimento contínuo da ferramenta.

3. Aplicações

No artigo sobre visão computacional de PEREIRA et al. (2018) [1] é dito que, a visão computacional é uma área bem-sucedida em tarefas como modelagem 3D, detecção de objetos em sequências de imagens, reconhecimento de digital, entre outras várias aplicações. Atualmente já é uma tecnologia muito presente na vida cotidiana, está contida nos aplicativos para reconhecimento facial, radares, leitura de códigos de barras e muitas outras coisas. De fato alguma destas tarefas são realmente complicadas para o olho humano, como determinar a velocidade de um carro somente ao olhar, ou instantaneamente dizer qual o valor atribuído a um código de barras.

Demonstrando os exemplos, vemos a evolução da tecnologia em radares de trânsito, estes possuem várias funções que podem depender da visão computacional, como já dito anteriormente, existem formas de fazer o cálculo da velocidade definindo pontos na imagem onde o carro detectado inicia e termina sua passagem pelo radar, a partir dai será necessário ter apenas a distância que aquele espaço possui e o tempo que o veículo levou para percorrer esta distância, uma outra aplicação pode ser a leitura de placa do veículo, que pode ser feita processando a imagem e, por fim, utilizando-se a tecnologia de reconhecimento ótico de caracteres, OCR, que vem do inglês Optical Character Recognition, para fazer a leitura dos caracteres e a partir dai pode-se vincular uma infração de excesso de velocidade, ou até reconhecer um veículo procurado por furto. Sobre os padrões de códigos de barra, quem utiliza serviços pelo internet banking, como pagamento de contas, já deve estar familiarizado com a utilização da câmera do smartfone para a leitura desse tipo de código, segundo a matéria publicada pela redação do site Canaltech [3], os códigos de barra seguem a mesma lógica da computação em geral: utilizam código binário para formar dados. As listras representam uma sequência de zero e um, sendo que o resultado dessa combinação é uma descrição exata do produto, tudo graças a uma base de dados consultada pelo equipamento leitor, que pode ser um computador ou smartfone, nesta sequência a parte preta representa os “uns” e a branca os “zeros”, assim ocorre o processamento e a definição de qual o valor do código aplicando a visão computacional.

4. Desenvolvimento

A ideia a ser desenvolvida e demonstrada será um contador de pessoas que passam por uma calçada, para isso será necessário fazer um processamento da imagem e definir pontos de interesse que serão utilizados e por fim o ponto que dirá se a pessoa passou subindo ou descendo na região de estudo, semelhante ao que foi explicado sobre os radares de trânsito na parte 3.

O primeiro passo é definir a linguagem de programação que será utilizada, a opção escolhida foi Python e para inciar foi feita a instalação do pacote contendo a biblioteca OpenCV, que pode ser encontrado diretamente no ambiente de desenvolvimento escolhido o Thonny, que é próprio para se utilizar com a linguagem Python.

Dando sequência é necessário abrir o vídeo a ser analisado pelo programa, este processo e todas as operações morfológicas são feitos pela biblioteca OpenCV, os seguintes passos foram seguidos:

1) Foi feita a modelagem do plano de fundo para uma melhor adaptação a pequenas mudanças de luminosidade e novos objetos sem movimento.

2) O próximo passo é passar o vídeo para a escala de cinza para realizar o seu processamento, de acordo com Nogueira (2016) [4], o objetivo geral é usar o número limitado de tons de cinza para preservar o máximo possível do contraste original, um frame do vídeo em escala de cinza pode ser visto no Anexo I.

3) Agora começa o processamento das imagens criando uma máscara e aplicando a limiarização, que terá como função remover as sombras da máscara pelo agrupamento dos pixels com intensidades de cinza mais parecidos, o limiar foi estabelecido de acordo com as características dos objetos que foram necessários de se isolar, o valor foi definido aumentando o limite inferior e superior até se encontrar o que deixou as identificações com maior precisão (transformando as partes mais acinzentadas da máscara para um tom mais escuro), estes valores foram 200 no limite inferior e 255 no limite superior, a comparação das imagens geradas pode ser observada nos Anexos II e III.

4) A seguir iniciam se as operações morfológicas, que foram vistas em aula [5], o primeiro passo é definir um kernel com forma elíptica para a convolução, este kernel é utilizado na operação de abertura, que terá como finalidade a redução do ruído, um frame resultado pode ser visto no Anexo IV. Fazendo a dilatação após a redução de ruído, os objetos têm sua área melhor definida, que pode ser visto no Anexo V, assim podendo ser melhor detectados pelo programa que será elaborado.

5) A última operação morfológica a ser aplicada é o fechamento, que fará a redução das degradações causadas pela operação de dilatação, que pode ser visto no Anexo VI.

6) Com todo o processamento de imagens feito agora técnicas de programação serão utilizadas para criar a visão computacional e detectar os objetos desejados, para demarcar os contornos uma função do OpenCV é utilizada, a partir daí o que passa a importar é o tamanho (área dentro do contorno) e o centro desta área, feito isso o ponto central será marcado por um círculo vermelho e o retângulo verde será desenhado de acordo com o tamanho e a posição do objeto,

7) Por fim as linhas representando os limites serão traçadas de forma que a central representa o local de contagem e a superior e inferior o offset, que é o local onde o ponto central passa a ser analisado e determina se ele contará uma subida ou descida. Um frame com o resultado final do que foi implementado pode ser visto no anexo VII e o código final com todas as operações e a lógica da programação aplicada pode ser visto no anexo VIII.

5. Conclusão

Dessa forma conclui-se que o trabalho demonstrativo mostrou resultados satisfatórios. Um total 18 pessoas transitaram pelo espaço de observação durante o período do vídeo e aconteceram 17 identificações, houve uma identificação a menos na descida. Sendo assim na subida foi obtido uma acurácia de 100% e na descida ela caiu para 90%, a única vez onde ocorreu falha na contagem foi quando duas pessoas passaram muito juntas, o que já era esperado visto que no processamento das imagens foi definido que pixels próximos fossem considerados um objeto só, essa definição pode ser vista com detalhes no passo 3 do item 4.

Na parte teórica pode ser absorvido que, a visão computacional engloba conceitos de diferentes áreas e sua evolução pode simplificar diversas tarefas, o que tornará mais fácil a análise dos dados e a produção nas empresas ou até mesmo a vida cotidiana, visto que se a tecnologia for aplicada de maneira correta, sua precisão será muito grande. Também deve se citar a grande importância das empresas continuarem a trabalhar com bibliotecas como OpenCV e desenvolvendo novos recursos ou até mesmo novas bibliotecas, pois a criação de ferramentas diminui a barreira de entendimento de quem trabalha e estuda melhores maneiras de aplicar tecnologias baseadas nas técnicas de visão computacional, evoluindo cada vez mais o mercado e descobrindo novas formas de aplicação, o que gera valor tanto para as empresas quanto para a sociedade que poderá usufruir da evolução tecnológica.

6. Referências

[1] PEREIRA, Jonas; LEMOS, Luan; MELHOR, Pierre. **Visão Computacional**. Concursos e Competições - SIC UFBA, Departamento de Ciência da Computação — Universidade Federal da Bahia (UFBA) — Salvador — BA — Brasil, 18 fev. 2018. Disponível em: https://medium.com/@06sicufba/artigo-sobre-visão-computacional-cd5d5ba81a00. Acesso em: 21 mar. 2021.

[2] OpenCV Team, **About**. Disponível em: https://opencv.org/about/. Acesso em: 22 mar. 2021.

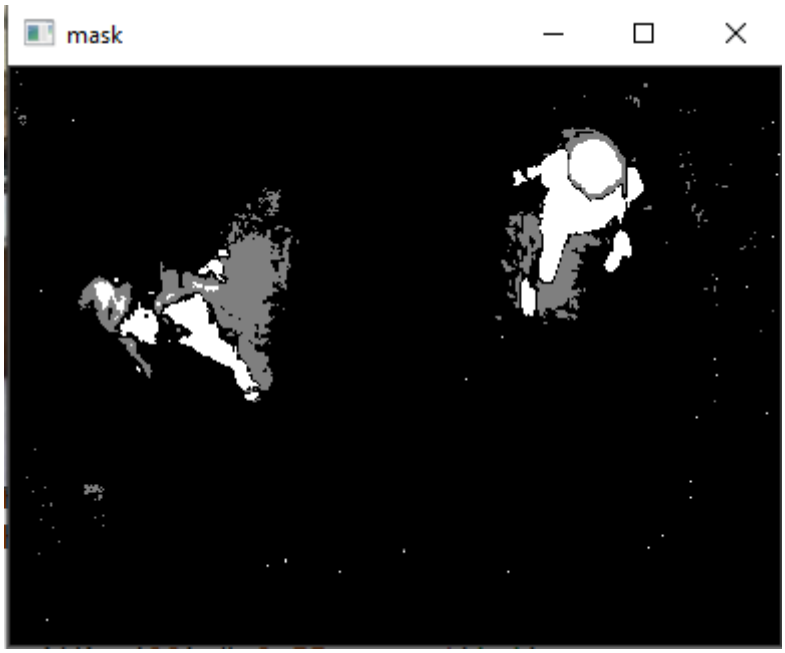
[3] Redação, Canaltech. **Entenda como funciona a leitura de códigos de barra.** Disponível em: https://canaltech.com.br/curiosidades/entenda-como-funciona-a-leitura-de-codigos-de-barra/. Acesso em: 23/03/2021.

[4] NOGUEIRA: Ricardo César de Almeida. *In*: NOGUEIRA: Ricardo César de Almeida. **Análise de Conversão de Imagem Colorida para Tons de Cinza Via Contraste Percebido**. Orientador: Prof. Dr. Carlos Alexandre Barros de Mello. 2016. TRABALHO DE GRADUAÇÃO (ENGENHARIA DA COMPUTAÇÃO) – UFPE — UNIVERSIDADE FEDERAL DE PERNAMBUCO, 2016. f. 46. Disponível em: https://www.cin.ufpe.br/~tg/2016-1/rcan.pdf. Acesso em: 22 mar. 2021.

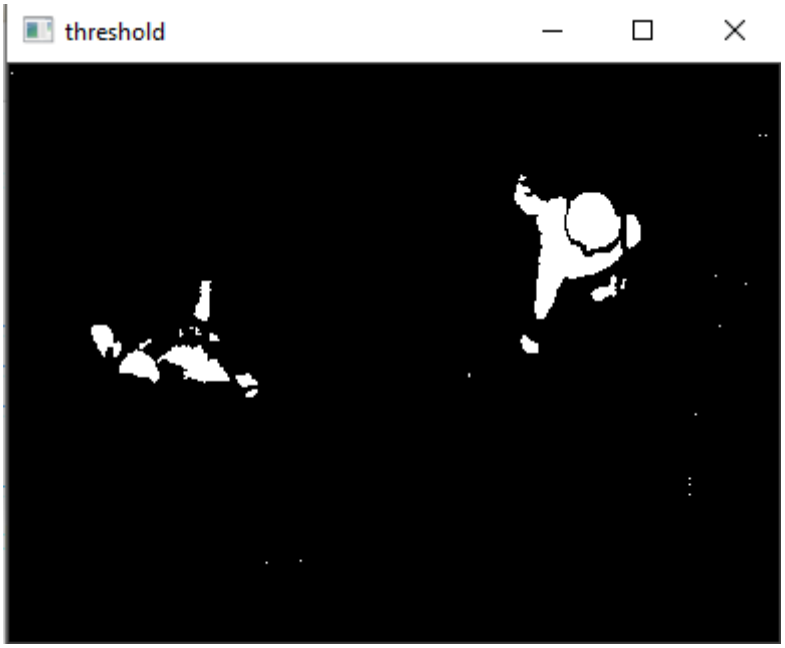
[5] NAKASHIMA, Natasha Sayuri Dias. **Notas de aula. Aula 13: Filtros Não-Lineares**, Universidade Federal de Itajubá - UNIFEI Campus de Itabira – MG, 15 de Março de 2021

7. Anexos

Anexo I: Frame do vídeo usado no projeto de demonstração em escala de cinza.

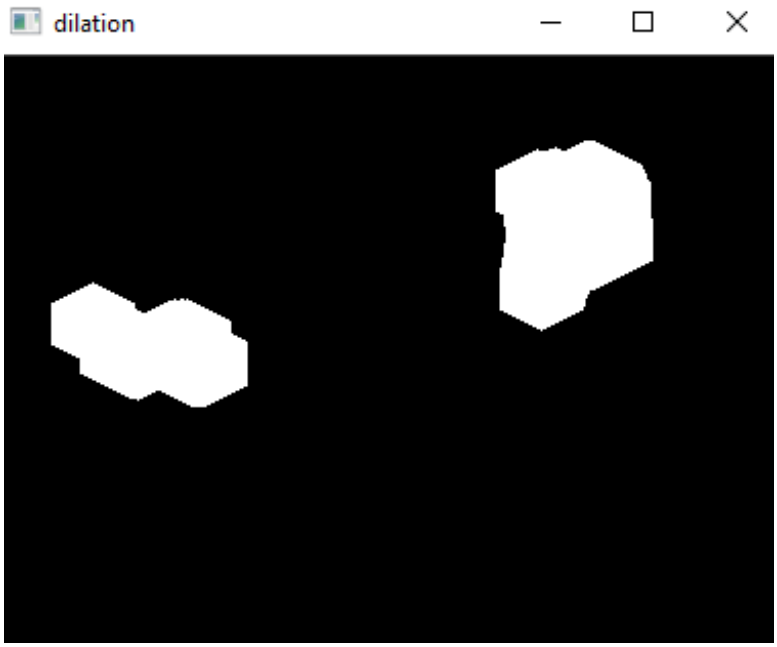


Anexo II: Frame do vídeo usado no projeto de demonstração com a máscara aplicada.

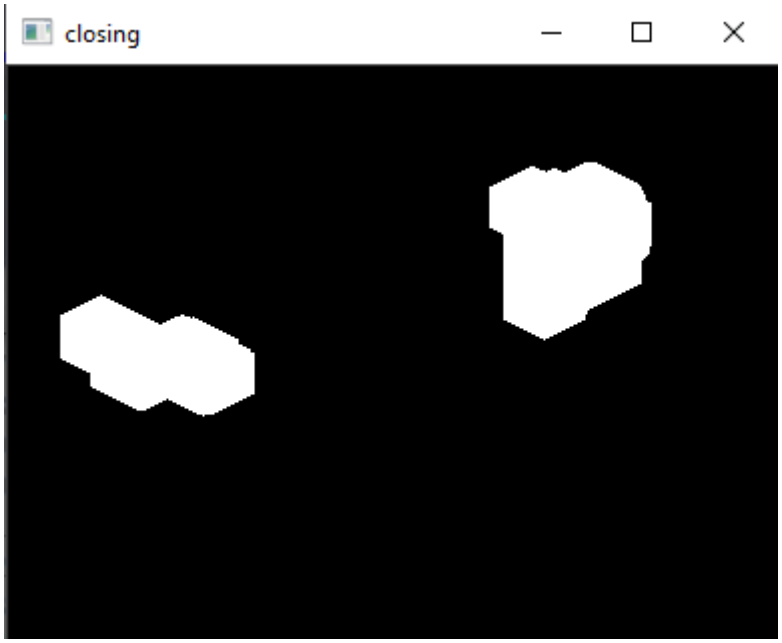


Anexo III: Processo de limiarização para remoção de sombras.

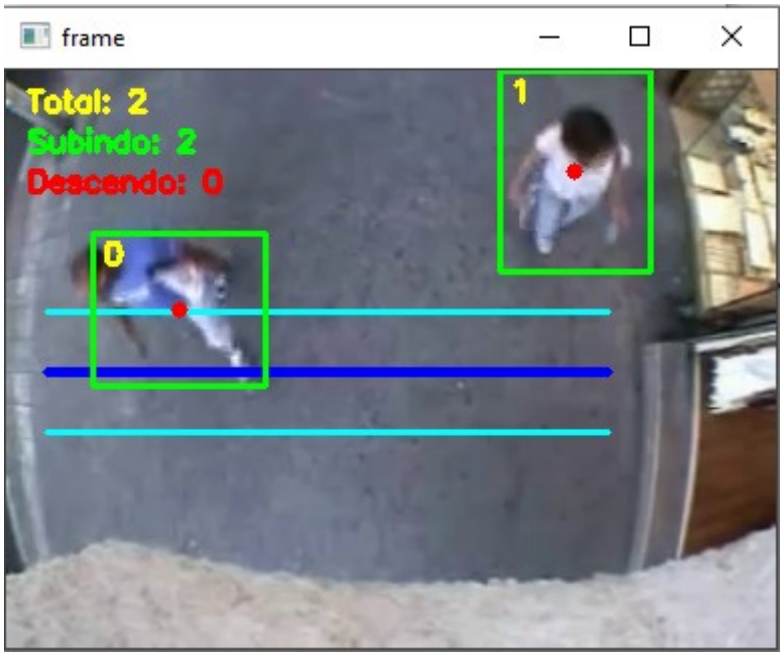
Anexo IV: Frame após a operação morfológica de abertura.



Anexo V: Frame após a operação morfológica de dilatação.



Anexo VI: Frame após a operação morfológica de fechamento.



Anexo VII: Resultado final do vídeo com as detecções.

```
import numpy as np
import cv2

#funcao para calcular o centro do contorno
def center(x, y, w, h):
    x1 = int(w/2)
    y1 = int(h/2)
    cx = x + x1
    cy = y + y1
    return cx,cy

#Opcoes das linhas
posL = 250
offSet = 50

xy1 = (120, posL)
xy2 = (720, posL)

######
#variaveis
detects = []
#contagem
total = 0
up = 0
down = 0

cap = cv2.VideoCapture('1.mp4')
fgbg = cv2.createBackgroundSubtractorMOG2()

while 1:
    ret, frame = cap.read()

    #converter imagem para cinza
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    #Gerar a mascara
    fgmask = fgbg.apply(gray)

    #remover sombras da mascara
    retval, th = cv2.threshold(fgmask, 200, 255, cv2.THRESH_BINARY)

    #remover noise da mascara
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
    opening = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel, iterations= 2)

    #diltar os objetos que restaram
    dilation = cv2.dilate(opening, kernel, iterations= 8)

    #fechar o reconhecimento de um objeto
    closing =cv2.morphologyEx(dilation, cv2.MORPH_CLOSE, kernel,
iterations= 8)

    #retirar contorno
    contours, hierarchy = cv2.findContours(closing, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    #gerar linhas
    cv2.line(frame, xy1, xy2,(255,0,0),3)
    cv2.line(frame,(xy1[0], posL - offSet), (xy2[0], posL - offSet),
(255,255,0),2)
    cv2.line(frame,(xy1[0], posL + offSet), (xy2[0], posL + offSet),
(255,255,0),2)

    i = 0 #id da contagem
    for cnt in contours:
        #calcular area
        (x, y, w, h) = cv2.boundingRect(cnt)
        area = cv2.contourArea(cnt)

        #limite do objeto
        if int(area) > 3000:
            #calcular centro
            centro = center(x, y, w, h)
            #gerar contorno e circulo
            cv2.circle(frame, centro, 4, (0,0,255), -1)
            cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0),2)
            #gerar codigo do objeto
            cv2.putText(frame, str(i), (x+5, y+15),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)

            #incrementar id
            if len(detects) <= i:
                detects.append([])
            if centro[1]> posL - offSet and centro[1] < posL + offSet:
                detects[i].append(centro)
            else:
                detects[i].clear()

            i += 1

    #limpar a lista
    if len(contours) == 0:
        detects.clear()
    else:
        for detect in detects:
```

```
            for (c,l) in enumerate(detect):
                #verificar se o objeto subiu
                if detect[c-1][1] < posL and l[1] > posL :
                    detect.clear()
                    up+=1
                    total+=1
                    cv2.line(frame,xy1,xy2,(0,255,0),5)
                    continue
                #verifica se o objeto desceu
                if detect[c-1][1] > posL and l[1] < posL:
                    detect.clear()
                    down+=1
                    total+=1
                    cv2.line(frame,xy1,xy2,(0,0,255),5)
                    continue
                #desenha a linha que segue o objeto
                if c > 0:
                    cv2.line(frame,detect[c-1],l,(0,0,255),1)

    #print(detects)

    #####
    #exibicoes
    cv2.putText(frame, "Total: " + str(total), (10,20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,255),2)
    cv2.putText(frame, "Subindo: " + str(up), (10,40),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0),2)
    cv2.putText(frame, "Descendo: " + str(down), (10,60),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),2)
    #video
    cv2.imshow("frame", frame)
    cv2.imshow("closing", closing)


    if cv2.waitKey(90) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Anexo VIII: Código completo do exemplo demonstrado.