Licenciatura Engenharia Informática e Multimédia

Processamento de Imagem e Visão

Trabalho Prático 2

Data: 14 fevereiro 2021

Realizado por:

Luís Fonseca - 45125
Rodrigo Correia - 45155

Grupo:4

Docentes:

Eng. Pedro Mendes Jorge

Índice de Conteúdos

List of Figures

Listings

1 Introdução

O segundo trabalho prático da unidade curricular de Processamento de Imagem e Visão foca-se no desenvolvimento de um algoritmo capaz de detetar, e classificar, zonas de imagem onde ocorreram movimentos de objectos.

Para tal recorremos às funcionalidades da biblioteca *OpenCV*, que nos permite fazer o tratamento do vídeo em tempo real indo de encontro aos objetivos do trabalho prático.

2 Objetivos

O objetivo deste trabalho é o desenvolvimento de um algoritmo capaz de encontrar e identificar qualquer tipo de movimento, neste caso, pessoas, carros e outros objetos em estudo(neste caso, no vídeo fornecido, um ciclista), localizando-as nas zonas de imagem onde ocorram movimento.

A realização deste algoritmo foi feita através da linguagem *Python* com suporte às suas bibliotecas, mais em concreto, a biblioteca *OpenCV2*, que nos irá dar muito jeito no que diz respeito a manipulação e processamento de imagem.

3 Estrutura do relatório e passos concretizados para o Algoritmo

Este relatório encontra-se divido em diferentes seções, estas seções seguem também as fases da realização do trabalho. Sendo assim este relatório está divido da seguinte forma:

- Deteção de pixeis ativos;
- Melhoramento da Imagem;
- Extração dos componentes;
- Identificação e Classificação dos componentes;
- Visualização de resultados.

4 Deteção de pixeis ativos

Começou-se o trabalho por ler o ficheiro alvo de estudo. De seguida inicializamos dois frames diferentes que são interpretados pela função *cv2.absdiff*, do *OpenCV2*, fazendo o seu módulo absoluto entre essas duas frames, com o objectivo de identificar os pixeis ativos entre os frames usando o método de subtração, ou seja identificar movimento. De salientar que este processo ocorre em tempo real enquanto o video está em reprodução.

```
cap = cv2.VideoCapture('camera1.mov')
ret, frame1 = cap.read()
ret, frame2 = cap.read()

while cap.isOpened():

  diff = cv2.absdiff(frame1, frame2)
```

Listing 1: Deteção de pixeis ativos

5 Melhoramento da Imagem

Inicalizamos o processo de melhoramento da imagem com o fim de obter os pixeis ativos de modo mais fiel possivel , uma vez que o ficheiro de video apresenta um pouco de ruido derivado da qualidade da câmera de vigilância.

Começamos por transformar a imagem numa escala de cinzentos, processo que facilita a sua análise na binarização. É ajustado de seguida o tamanho da tela, pois utilizando um tamanho inferior ao utilizado, visto que diminui-se a resolução do mesmo, ficando assim com menos ruido.

Aplicamos agora um "medianblur", método que analisa a vizinhança do pixel alvo e substitui pela sua mediana, obtendo assim uma imagem mais suave com menos contraste e removendo grande parte do ruído da imagem.

Estando a imagem melhorada, fazemos a binarização com recurso a um "treshold" binário. Ao ser utilizado um threshold binário ficamos com uma imagem em que os pixeis ativos são representados a branco e tudo o resto a preto.

Finalmente concluimos o processo por aplicar o operador morfológico da dilatação, com objetivo de realçar um pouco os pixeis ativos tornando-os mais uniformes.

Na figura a seguir, podemos verificar o processo de melhoramento da imagem, com todos os operadores morfológicos aplicados.

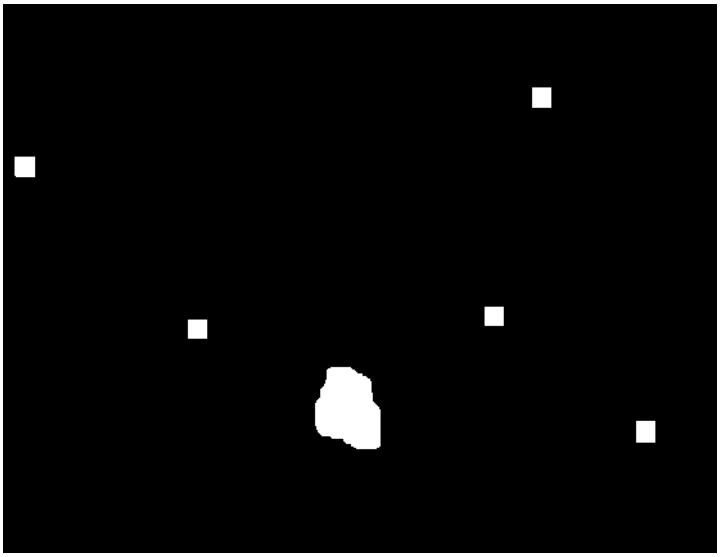

Figure 1: Binarização da imagem, com os píxeis ativos, em preto e branco

A implementação desta seção consiste no seguite: aplicamos tons de cinzento às diferentes imagens, sendo mais fácil aplicar binarização. De seguida, usamos o método *resize()* que como já foi referido, conseguimos alterar a resolução, ficando mais pequeno, retirando parte do ruído. Antes de aplicar um Threshold, usamos o método *mediaBlur* para suavizar a imagem, sendo mais fácil detetar pixeis ativos, e melhorar o ruído da imagem. No final usamos o método *dilate* para aperfeiçoar os pixeis que a imagem encontra.

```
1  gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
2  gray = cv2.resize(gray,None,fx=0.8,fy=0.8,interpolation = cv2.
     INTER_LINEAR)
3  frame1 = cv2.resize(frame1,None,fx=0.8,fy=0.8,interpolation = cv2
     .INTER_LINEAR)
4  blur = cv2.medianBlur(gray, 5)
5  _, thresh = cv2.threshold(blur, 10, 255,cv2.THRESH_BINARY)
6  dilated = cv2.dilate(thresh, None, iterations=7)
7  cv2.imshow("as", dilated)
```

Listing 2: Melhoramento da imagem

6 Extração dos componentes

Para que possamos com mais facilidade, detetar quais as regiões com um maior conjunto de pixeis, foi necessário definir contornos. Sempre que existe um aglomerado de pixeis, consoante uma determinada área, iremos detetar os diferentes contornos, e consoante a sua área, identificar os diferentes tipos de objectos, quer seja uma pessoa, um carro ou um ciclista.

Em python, recorremos ao método *findContour*, permitindo ser mais fácil detetar os diferentes contornos, e passando os seguintes argumentos: *cv2.RETR_TREE* que permite retornar todos os contornos existentes, e o argumento *cv2.CHAIN_APPROX_SIMPLE* que permite remover todos os pontos redundantes, comprimindo o contorno.

No final, usamos a condição *cv2.contourArea(contour)* , ou seja, caso seja verificamos qual a área do contorno, e caso ela seja menor que 700, continuamos o processo de encontrar diferentes contornos.

```
1  contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
2  for contour in contours:
3    if cv2.contourArea(contour) < 700:
4      continue
```

Listing 3: Deteção dos contornos

7 Identificação e Classificação dos componentes

Depois de identificados os diferentes contornos, passado na imagem binarizada, passamos para a identificação dos diferentes componentes, neste um PESSOA, um CARRO e OUTRA. Consoante a sua "bounding box", é identificado os diferentes tipos de componentes.

Em python, par a criação das bounding boxes, recorremos ao método *boundingRect*. De seguida, para detetar consoante o tipo de componente que estava presenta no vídeo, foram criadas duas variáveis, que consistem na altura e largura das bounding boxes. Ambas as variáveis lêem os respetivos valores das diferentes bounding boxes.

No caso para distinguir os diferentes componentes na imagem, verificamos a condicao se altura(h) é maior que a largura (w). Caso seja verdadeira esta condição, então identificamos esse componente como sendo uma PESSOA. No entao caso, a altura(h) seja menor ou igual à largura (w), identificamos esse componenete como sendo um CARRO. Caso a primeira condição não seja verdadeira, o tipo de componente é classificado como sendo OUTRO.

Os métodos usados para identificação dos diferentes componentes foram os seguintes: o método *cv2.rectangle* que desenha um retângulo à volta dos diferentes componentes, recebendo como argumentos, uma frame, a posição que é detetada (neste caso as coordenadas x e y), a posição final (neste caso será a largura e altura do componente), atribuir uma cor diferente a cada retângulo desenhado, e a espessura desse retângulo. Foi também usado o método *cv2.putText* que coloca texto(funcionando como uma legenda) para identificar os diferentes componentes.

```
1     x, y, w, h = cv2.boundingRect(contour)
2     largura = x+w
3     altura = y+h
4     if(h>w*1.30):
5       #pessoa
6       cv2.rectangle(frame1, (x, y), (largura,altura), (255, 0, 0),
    2)
7       cv2.putText(frame1, "Pessoa" ,(x, y), cv2.
    FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255),)
8     elif(h<=w):
9       #carro
10      cv2.rectangle(frame1, (x, y), (largura,altura), (0, 255, 0),
    2)
11      cv2.putText(frame1, "Carro" ,(x, y), cv2.FONT_HERSHEY_SIMPLEX
    , 1, (255, 255, 255),)
12    else:
13        #outro
14      cv2.rectangle(frame1, (x, y), (largura,altura), (0, 0, 255),
    2)
15      cv2.putText(frame1, "Outro", (x, y), cv2.FONT_HERSHEY_SIMPLEX
    ,1, (0, 0, 255),)
```

Listing 4: Identificação e Classificação dos componentes

8 Visualização de resultados

Para finalizar a implementação deste algoritmo, em python, usamos o método *cap.read()*, permitindo ler as frames do vídeo, e usando o comando *cv2.imshow* que permite visualizar o vídeo, depois de ter sido implementado o algoritmo de deteção de movimento.

```
  cv2.imshow("feed", frame1)
  frame1 = frame2
  ret, frame2 = cap.read()
  if not ret:
    break
  if cv2.waitKey(1) == 27:
    break
cv2.destroyAllWindows()
```

Listing 5: Leitura do vídeo após ter sido passado o algoritmo de deteção de vídeo

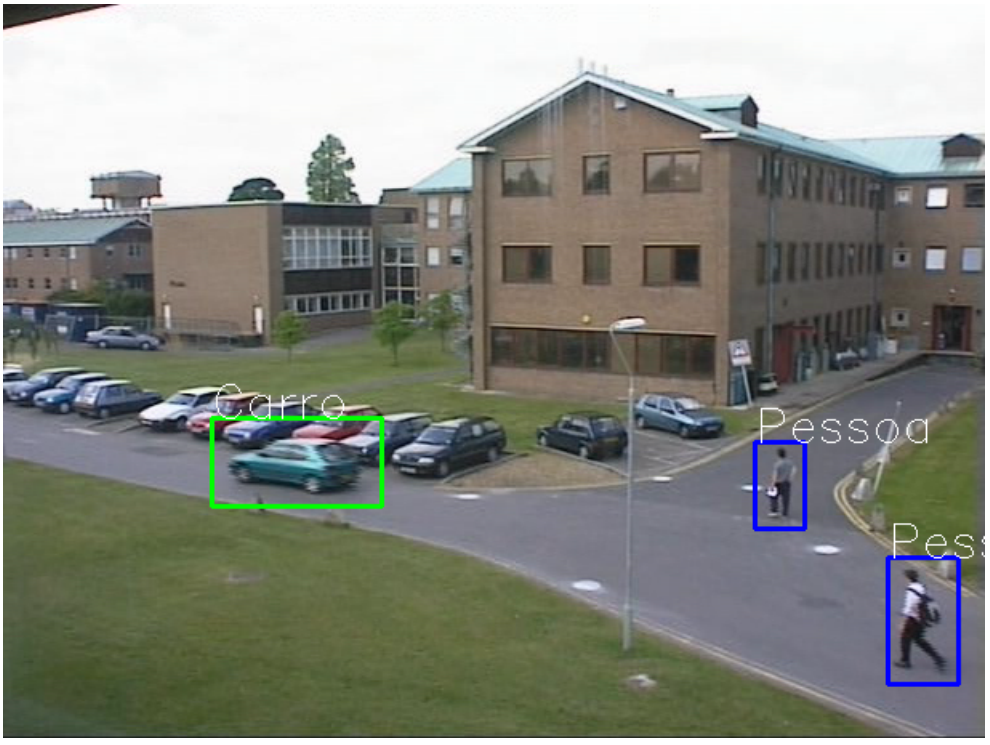

Figure 2: Visualização final dos resultados, com bondingBoxes e identificação das mesmas sobre os pixeis ativos.

9 Conclusões

De modo a recapitular todos os aspetos sobre os quais o algoritmo desenvolvido debruça serão referidos os diferentes momentos da implementação.

O processamento inicial da imagem é de extrema importância, uma vez que toda a restante implementação se baseia neste processamento.

Após este processamento inicial os métodos desenvolvidos focaram-se, principalmente, sobre a análise e processamento das regiões ativas, sendo de destacar a importância das mesmas para o algoritmo. Destas regiões resultam as bounding boxes que permitem o desenho das trajetórias, bem como a identificação e classificaçã das regiões.

Deste modo é possível afirmar que o algoritmo desenvolvido, quando utilizado como um todo, obtêm resultados satisfatórios, existindo, no entanto incoerências relativamente aos identificadores, verificando-se em certas ocasiões erros, como por exemplo, o facto de alguns valores serem idênticos, neste caso da altura e da largura, a certo ponto terem valores iguais, criando "salto" entre o tipo de componenete identificado, sendo este erro, comun quando estamos a tentar classificar uma PESSOA, alterando o seu nome e bonding box para o componenete OUTRO . Este factor deve-se ao facto de surgirem regiões ativas em zonas onde estão ou estiveram anteriormente, regiões ativas, e estando as coordenadas destas mesmas regiões guardadas, o identificador acaba por ser o mesmo.

Outra exeção é nos casos onde os movimentos são curtos e/ou de amplitude muito reduzida, como no caso do veículo que está estacionado inicia a manobra de marcha átras. O deslocamento é feito muito devagar,portanto o algoritmo ao fazer a subtração entre frames encontra poucas regiões ativas, não identificando corretamente.

Outra exeção é nos casos onde os movimentos são curtos e/ou de amplitude muito reduzida, como no caso do veículo que está estacionado inicia a manobra de marcha átras.
O deslocamento é feito muito devagar,portanto o algoritmo ao fazer a subtração entre frames encontra poucas regiões ativas, não identificando corretamente.

Nos restantes aspetos o algoritmo apresenta resultados satisfatórios, sendo o erro acima o único detetado durante a implementação e teste do programa realizado.

É ainda de destacar a importância deste trabalho prático no que diz respeito à familiarizaçãao com algoritmos de deteção e processamento de regiões ativas. Estes algoritmos são cada vez mais utilizados em aplicações de sistemas de videovigilância e realidade aumentada, sendo aplicáveis onde seja necessário o processamento de imagem em tempo real, como em análise de videovigilância e desportos que incluam a aplicação de novas tecnologias como o ténis, basket, e mais recentemente o futebol.

10 Bibliografia

- Slides fornecidos pelo docente, da Unidade Curricular de Processamento de Imagem e Visão
- Documentação OpenCV - https://docs.opencv.org/
- Informações uteis OpenCv - https://stackoverflow.com/questions/tagged/opencv