2ºTrabalho Laboratorial

January 23, 2022

Instituto Superior de Engenharia de Lisboa

Ano Lectivo: 2021/2022

Processamento de Imagem e Visão

2º Trabalho Laboratorial - Deteção de movimento para Tracking de objetos

Docentes:

Pedro Mendes Jorge e João Costa

Desenvolvido por:

Aluno

Número

Ana Sofia Oliveira

39275

Índice

Introdução

Este trabalho prático irá ser desenvolvido no ambito da unidade curricular de Processamento de Imagem e Visão e tem como principal objectivo construir um sistema capaz de identificar pessoas e carros em movimento num vídeo. Para isto irá ser necessário identificar as zonas em que existe movimento e distinguir pessoas e carros dos restantes objectos que possam ser apresentados no vídeo.

Irei apresentar um sistema capaz de implementar diversas técnicas de processamento de imagem, lecionadas no decorrer da unidade curricular, tais como subtração de fundo, conversão de imagens para níveis de cinzento, binarização de imagens, melhoramento de imagens, extração de

componentes conexos, extração de propriedades e classificação de objectos. Todos estes conceitos irão ser abordados no decorrer deste documento.

Serão apresentados testes e resultados por cada técnica de processamento em foco, bem como os testes e resultados finais. Todos estes resultados serão apresentados como imagens, excepto na extração de propriedades em que, para além das imagens, será complementado com um ficheiro .txt com as propriedades de todos os objectos analisados na imagem corrente.

Durante a implementação deste trabalho serão utilizados métodos de processamento de imagem, disponibilizados pela biblioteca Open CV, métodos de visualização de dados e de apresentação gráfica, disponibilizados pela biblioteca Matplotlib.pyplot, e métodos para cálculo matemático, disponibilizados pela biblioteca Numpy.

[1]:
```
import cv2
import numpy as np
import math

path = "video/"
```

Desenvolvimento

Um sistema de deteção de movimento é um sistema capaz de detetar os objectos em movimento e classificá-los de acordor com a sua categoria. Este sistema verifica uma determinado vídeo analizando-o frame-by-frame e classificandos os objectos em cada uma delas. No meu caso pertendia construir um detetor de movimento capaz de distiguir carros, pessoas e outros objectos em movimento. Para cada frame, pretendia obter frames de saída com todos os objectos identificados classificados.

Desta forma, podemos decrever o seguinte diagrama geral do sistema a implementar:

O sistema deve contemplar os seguintes processos:

Leitura frame-by-frame do vídeo;

Estimação de fundo;

Deteção de pixeis ativos;

Melhoramento da imagem;

Extração de propriedades dos objectos da imagem;

Classificação dos objectos da imagem.

[Adicional] Segmentação (tracking) dos objectos entre frames.

Cada uma destas técnicas será descrita como um bloco no sistema implementado. No decorrer deste documento irei detalhar todas as técnicas acima indicadas. Podemos agora detalhar o sistema conforme apresentado no diagrama abaixo.

1. Estimação de Fundo

A estimação de fundo é um técnica utilizada em sistemas de vídeo para distinguir os objectos em movimento daqueles que se mantém estáticos.

O fundo de um vídeo, embora possa ser estático, pode ser influenciado por diversos fatores externos, como a alteração da luminosidade ao longo do tempo. Isto faz com que, quando mantemos sempre o mesmo fundo sem poderar estes possíveis fatores, em algumas situações, possamos estar a identificar partes do fundo como movimento, indevidamente.

Assim, para colmatar estes fatores, o fundo depende em parte da última frame lida e torna-se, também ele, dinâmico ao longo do tempo. Para isto, define-se que caso exita movimento na última frame lida o novo fundo é igual ao fundo anterior. No entanto, nas zonas em que não existe movimento na última frame lida, o fundo é composto por parte da frame lida e parte do antigo fundo, na mesma zona. O fator que define se queremos manter mais informação mais recente - maior preponderancia da última frame lida - ou mais antiga - maior preponderancia do fundo, é o fator α definido entre [0, 1].

A fórmula geral da estimação de fundo é a seguinte:

Pixéis de movimento:

$|I_n(r,c) + B_n(r,c)| > T_n(r,c)$

Cálculo do fundo:

$$B_{n+1}(r,c) = \begin{cases} B_n(r,c), & \text{se (r,c) pixel movimento} \\ \alpha B_n(r,c) + (1-\alpha) I_n(r,c), & \text{se caso contrario} \end{cases}$$

Para detetar os pixeis de movimento construí a função deteta_movimento e para calcular o fundo construí a função estima_fundo. A função deteta_movimento recorre à função absdiff para calcular a diferença entre o fundo e a frame corrente, e guarda as posições onde essa diferença é superior ao threshlod definido. A função estima_fundo recorre à função addWeighted para calcular o novo fundo.

O threshold escolhido para a deteção do movimento foi definido após alguns testes. Caso o threshold fosse muito elevado existam objectos de interesse cujo movimento não era identificado devido à baixa variação de cor entre o fundo e a frame de análise. Caso o threshold fosse muito baixo, qualquer variação era identificada como pixel de movimento. Assim, este threshold foi escolhido com base nos valores de teste que melhor serviam o propósito do trabalho.

O valor de α foi escolhido de forma a que o fundo mantivesse as mesmas caracteristicas e fosse pouco influenciado pela frame corrente. Caso contrário qualquer movimento entre o fundo e a frame iria durar mais tempo até que fosse atenuado na imagem, fazendo com que o peso das frames anteriores no fundo fosse maior.

[2]:
```
def deteta_movimento(fundo, imagem_seguinte, threshold):

    diff = cv2.absdiff(fundo, imagem_seguinte)
    movimento = diff > threshold

    return movimento


def estima_fundo(fundo, imagem_seguinte, movimento, alfa = 0.9):

    pixeis_fundo = fundo*movimento
```

```
    pixeis_movimento = cv2.addWeighted(fundo, alfa, imagem_seguinte, (1-alfa),␣
↪0)*np.invert(movimento)

    fundo = cv2.add(pixeis_fundo, pixeis_movimento)

    return fundo
```

[3]:
```
capture = cv2.VideoCapture(path + "camera1.mov")
fps = capture.get(cv2.CAP_PROP_FPS)
total_frames = capture.get(cv2.CAP_PROP_FRAME_COUNT)

_, background = capture.read()
_, frame = capture.read()
frameCount = 1

while capture.isOpened():

    _, next_frame = capture.read()
    frameCount = frameCount + 1

    # Deteção de movimento e estimação do fundo
    movimento = deteta_movimento(frame, next_frame, 20)
    background = estima_fundo(background, next_frame, movimento, 0.9)

    cv2.imshow("Fundo", background)

    frame = next_frame
    key = cv2.waitKey(int(1000/fps))

    if(key == 27 or frameCount == total_frames-1):
        cv2.destroyAllWindows()
        capture.release()
```

2. Deteção de pixeis ativos

   A deteção de pixeis ativos é essencial neste sistema porque é o primeiro passo para nos peritir identificar regiões onde existiu movimento e, mais tarde, classificar essas regiões.

   No bloco anterior já tinham sido calculados os pixeis onde existiu movimento, que correspondem aos pixéis ativos na imagem, pelo que, de forma a conseguirmos apresentar graficamente esta informação criei a função obtem_pixeis_ativos, que multiplica todos os pixeis da matriz de movimento por 255.

[4]:
```
def obtem_pixeis_ativos(imagem_movimento):
    pixeis_ativos = imagem_movimento.astype(np.uint8)*255

    return pixeis_ativos
```

[5]:
```
capture = cv2.VideoCapture(path + "camera1.mov")
fps = capture.get(cv2.CAP_PROP_FPS)
total_frames = capture.get(cv2.CAP_PROP_FRAME_COUNT)

_, background = capture.read()
_, frame = capture.read()
frameCount = 1

while capture.isOpened():

    _, next_frame = capture.read()
    frameCount = frameCount + 1

    # Deteção de movimento e estimação do fundo
    movimento = deteta_movimento(frame, next_frame, 20)
    background = estima_fundo(background, next_frame, movimento, 0.9)

    # Pixeis ativos
    pixeis_ativos = obtem_pixeis_ativos(movimento)

    cv2.imshow("Pixeis Ativos", pixeis_ativos)

    frame = next_frame
    key = cv2.waitKey(int(1000/fps))

    if(key == 27 or frameCount == total_frames-1):
        cv2.destroyAllWindows()
        capture.release()
```

3. Melhoramento de Imagem

   Com a identificação dos pixeis ativos existiam alguns pixeis que, para o propósito do trabalho não interssavam ser considerados e algumas regiões que, embora estivessem separadas, pertenciam ao mesmo objecto. Assim, foi necessário usar técnicas de melhoramento de imagem de forma a definir as áreas de interesse.

   Como a frame de pixeis ativos continha 3 canais de cor e de forma a facilitar o processamento da imagem foi realizada a binarização da mesma antes de prosseguir com o melhoramento da imagem.

   Conforme indicado anteriormente, o objectivo do trabalho é a identificação de pessoas e carros. Estes objectos, à primeira vista, no vídeo de análise têm formas geométricas gerais muito identicas pelo que o elemento estruturante usado para o melhoramento da imagem foi o cv2.MORPH_RECT. Dependendo da operação morfológica usada, este elemento irá alterar o seu tamanho.

   Após a escolha do elemento estruturante - retangulo - foi necessário escolher os operadores morfológicos a usar e qual a sua ordem. Assim, a sequência aplicada no âmbito do trabalho foi a seguinte:

1. Fecho

   Tamanho do elemento estruturante: (3,3)

Objectivo: Remover pixeis activos cujo com tamanho igual ao inferior ao tamanho do elemento estruturante para que fosse considerados para a extração de propriedades.

2. Dilatação
   Tamanho do elemento estruturante: (8,8)
   Objectivo: Aumentar as zonas onde foi identificado movimento para facilitar a identificação dos objectos.

3. Abertura
   Tamanho do elemento estruturante: (15,10)
   Objectivo: Juntar áreas que após a dilatação ainda pudesses estar separadas mas que correspondiam ao mesmo objecto.

O tamanho dos elementos estruturantes aplicados são os que, após alguns testes, melhor cumpriram o objectivo do trabalho.

[6]:
```
def binarizaImagem(pixeis_ativos):
    kernel_median = (5,5)
    next_frame_g = cv2.cvtColor(pixeis_ativos, cv2.COLOR_BGR2GRAY)
    next_frame_g = cv2.blur(next_frame_g, kernel_median)
    th, binary = cv2.threshold(next_frame_g, 75, 255, cv2.THRESH_BINARY)

    return th, binary
```

[7]:
```
def melhoraImagem(pixeis_ativos):

    _, binary = binarizaImagem(pixeis_ativos)

    elementoEstruturante = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
    fecho = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, elementoEstruturante)

    elementoEstruturante = cv2.getStructuringElement(cv2.MORPH_RECT, (8,8))
    dilatacao = cv2.dilate(fecho, elementoEstruturante, iterations = 3)

    elementoEstruturante = cv2.getStructuringElement(cv2.MORPH_RECT, (14,7))
    abertura = cv2.morphologyEx(dilatacao, cv2.MORPH_OPEN, elementoEstruturante)

    return abertura
```

[8]:
```
capture = cv2.VideoCapture(path + "camera1.mov")
fps = capture.get(cv2.CAP_PROP_FPS)
total_frames = capture.get(cv2.CAP_PROP_FRAME_COUNT)

_, background = capture.read()
_, frame = capture.read()
frameCount = 1

while capture.isOpened():
```

```
    _, next_frame = capture.read()
    frameCount = frameCount + 1

    # Deteção de movimento e estimação do fundo
    movimento = deteta_movimento(frame, next_frame, 20)
    background = estima_fundo(background, next_frame, movimento, 0.9)

    # Pixeis ativos
    pixeis_ativos = obtem_pixeis_ativos(movimento)

    # Melhoramento da imagem
    melhorada = melhoraImagem(pixeis_ativos)
    cv2.imshow("Pixeis Ativos - Melhoramento de Imagem", melhorada)

    frame = next_frame
    key = cv2.waitKey(int(1000/fps))

    if(key == 27 or frameCount == total_frames-1):
        cv2.destroyAllWindows()
        capture.release()
```

4. Extração de Propriedades

   Após o melhoramento da imagem, é possível extraír caracteristicas ou propriedades das regiões ativas. Assim, foi efetuada a análise da área, largura e altura destas regiões e verificadas quais as caracteristicas que melhor diferenciavam os objectos a classificar. Para isso, primeiro foi guardada a informação das caracteristicas extraídas de várias imagens num ficheiro .txt que para cada imagem identificava o objecto e as suas caracteristicas. Após esta extração, foram mantidas apenas as imagens em que as caracteristicas dos objectos minimizavam as caracteriticas iniciais de deteção. Por exemplo, se quisermos saber a partir de que área devemos considerar o objecto para classificação e, no vídeo, existem frames onde o objecto tem menores porporções, então precisamos de considerar a frame onde o objecto tem menor tamanho para extraír o menor valor da área que o objecto pode ter.

   Desta forma foram analisadas as seguintes frames e as propriedades dos objectos nela identificados:

Frame

Objecto

Área

Largura

Altura

1445

1 - Pessoa

3618.5

53

87

1445

2 - Pessoa

2024.5

43

57

1445

3 - Carro

4402.0

92

57

Frame

Objecto

Área

Largura

Altura

1860

1 - Pessoa

1797.0

41

54

Frame

Objecto

Área

Largura

Altura

2342

1 - Pessoa

2138.0

42

62

Frame

Objecto

Área

Largura

Altura

2931

1 - Pessoa

1998.5

49

49

Frame

Objecto

Área

Largura

Altura

2949

1 - Carro

3984.5

81

57

A implementação da extração de propriedades foi realizada no método extracaoPropriedades e irá ser detalhado com maior pormenor no próximo tópico.

[9]:
```
def obtemArea(contorno):
    return round(cv2.contourArea(contorno), 2)

def obtemCentroide(contorno):
    momentos = cv2.moments(contorno)
    centro_x = round(momentos['m10']/momentos['m00'])
    centro_y = round(momentos['m01']/momentos['m00'])

    return (centro_x, centro_y)
```

5. Classificação

   É nesta fase que o processo de classificação própriamente dito inicia, sendo identificadas as caracteristicas (features) de cada objecto a classificar pelo sistema. Após a análise das propriedades nas frames anteriormente apresentadas foram retiradas as seguinte ilações:
   Área de pessoas compreendida entre [1500, 3500] pixéis;
   Área de carros superior a 3500 pixéis;
   Carros têm maior área que as Pessoas;

Largura de pessoas superior a 40 pixéis e Altura de pessoas superior a 50 pixéis;
Largura de carros superior a 81 pixéis pixéis e Altura de carros superior a 57 pixéis;
Pessoas têm Altura > Largura;
Carros têm Largura > Altura;
Objectos a classificar têm de Largura entre [Largura mínima Pessoa; Largura Máxima Carro] e Altura entre [Altura mínima Carro; Altura máxima Pessoa];
Objectos a classificar têm área superior à área mínima da Pessoa (1500 pixéis).

Através das ilações apresentadas acima, complementei o método extraçãoPropriedades para que apenas verificasse as propriedades dos objectos com largura, altura e área superiores aos valores apresentados acima, e desenvolvi o método classificaObjecto para que classificasse os objectos de acordo com as diferenças aprensentadas acima.

[10]:
```python
def classificaObjecto(area, largura, altura):
    if (altura>=largura) and (area > 1500):
        nome = 'Pessoa'
        cor = (0, 0, 255)

    elif (area >= 3500):
        nome = 'Carro'
        cor = (0, 255, 0)

    else:
        nome = 'Objecto'
        cor = (255, 0, 0)

    return nome, cor
```

[11]:
```python
def extracaoPropriedades(contornos, frameCount, objectos_encontrados, ID = 1):
    for cnt in contornos:
        # Calculate area and remove small elements
        area = obtemArea(cnt)
        posx, posy, largura, altura = cv2.boundingRect(cnt)

        if ((largura,altura) >= (40,50) and (area > 1500)):

            centroide = obtemCentroide(cnt)
            nome, cor = classificaObjecto(area, largura, altura)

            propriedades = [ID, {"nome": nome,
                                 "area": area,
                                 "centroide": centroide,
                                 "posx": posx,
                                 "posy": posy,
                                 "largura": largura,
                                 "altura": altura,
                                 "cor": cor},
                                frameCount]
```

```
            ID = ID + 1
            objectos_encontrados.append(propriedades)

    return objectos_encontrados, ID
```

6. Tracking de Objectos

   Após a classificação dos objectos por frame é necessário saber quais os objectos que já foram identificados em frames anteriores e se ainda se mantém no plano de imagem. Estes objectos deverão ser atualizados porém não devem ser considerados como novos objectos nos sistema.

   Este processo foi implementado no método trackObjects que compara os objectos detetados em frames anteriores com os objectos detetados na frame corrente. Caso o objecto já tenha sido detetado mantém a referência do ID previamente detetado, atualizando apenas as suas propriedades e a referência para a última frame onde foi observado. Caso o objecto não tenha sido detetado em frames anteriores adiciona-o à lista de objectos em tracking. Cada objecto mantém-se em tracking até que deixe de ser identificado em 5 frames seguidas. Caso isto aconteça, o objecto deixa de ser "acompanhado".

[12]:
```
def updateObjectOnTrack(old_object, new_object):
    # ID antigo, propriedades novas e frameID novo
    updated_object = [old_object[0], new_object[1], new_object[2]]

    return updated_object
```

[13]:
```
def trackObjects(tracking_objects, frame_objects, frameAtual):
    segment_objects = tracking_objects.copy()
    for frame_object in frame_objects:
        object_exists = False
        for track_object in tracking_objects:
            obj_centroid = frame_object[1]["centroide"]
            trk_centroid = track_object[1]["centroide"]

            distancia = math.hypot(trk_centroid[0] - obj_centroid[0],␣
 ↪trk_centroid[1] - obj_centroid[1])

            if(distancia < 10):
                # object in track
                object_exists = True
                track_object = updateObjectOnTrack(track_object, frame_object)
                continue

        if(not object_exists):
            # new object on track
            segment_objects.append(frame_object)

    tracking_objects = []
```

```
    # Valida se os objectos em track foram identificados nas últimas 5 frames
    for objecto in segment_objects:
        last_frame_seen = objecto[2]
        if(frameAtual - last_frame_seen < 5):
            tracking_objects.append(objecto)

    return tracking_objects
```

[14]:
```
def desenhaObjectos(tracking_objects, frame):
    contoured_frame = frame.copy()
    for objecto in tracking_objects:
        cv2.rectangle(contoured_frame, (objecto[1]["posx"],␣
↪objecto[1]["posy"]), (objecto[1]["posx"] + objecto[1]["largura"],␣
↪objecto[1]["posy"] + objecto[1]["altura"]), objecto[1]["cor"], 2)
        cv2.putText(contoured_frame, str(objecto[0]) + " " +␣
↪str(objecto[1]["nome"]), (objecto[1]["posx"], objecto[1]["posy"] - 7), cv2.
↪FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), thickness=1)

    return contoured_frame
```

Bateria de Testes Finais

[15]:
```
def MotionDetection(inVideo):

    capture = cv2.VideoCapture(inVideo)

    fps = capture.get(cv2.CAP_PROP_FPS)
    total_frames = capture.get(cv2.CAP_PROP_FRAME_COUNT)
    print("FPS:", fps)
    print("Total Frames:", total_frames)

    _, background = capture.read()
    _, frame = capture.read()

    frameCount = 1
    tracking_objects = []

    while capture.isOpened():

        _, next_frame = capture.read()
        frameCount = frameCount + 1

        # Deteção de movimento e estimação do fundo
        movimento = deteta_movimento(frame, next_frame, 20)
        background = estima_fundo(background, next_frame, movimento, 0.9)
#         cv2.imshow("background", background)
#         cv2.imshow("Next Frame", next_frame)
```

```
        # Pixeis ativos
        pixeis_ativos = obtem_pixeis_ativos(movimento)
#         cv2.imshow("Pixeis ativos", pixeis_ativos)

        # Melhoramento da imagem
        melhorada = melhoraImagem(pixeis_ativos)
#         cv2.imshow("Melhorada", melhorada)

        # Extração de regiões
        contoured = next_frame.copy()
        contours, hierarquia = cv2.findContours(melhorada, cv2.RETR_TREE, cv2.
↪CHAIN_APPROX_SIMPLE)

        # Extração de propriedades e classificação de objectos
        frame_objects = []
        frame_objects, ID = extracaoPropriedades(contours, frameCount,␣
↪frame_objects)

        # Tracking objects
        tracking_objects = trackObjects(tracking_objects, frame_objects,␣
↪frameCount)

        # Desenha objectos
        contoured_frame = desenhaObjectos(tracking_objects, next_frame)
        cv2.imshow("Contoured", contoured_frame)

        if(frameCount == 2949 or frameCount == 2931 or frameCount == 2342 or␣
↪frameCount == 1860 or frameCount == 1445 or frameCount == 1443):
            imagem_numerada = next_frame.copy()
            cv2.imwrite("imgs/pixeis_ativos/frame_" + str(frameCount) + ".jpg",␣
↪pixeis_ativos)
            cv2.imwrite("imgs/melhoramento/frame_" + str(frameCount) + ".jpg",␣
↪melhorada)

            file_name = "imgs/propriedades/frame_" + str(frameCount) + ".txt"
            file = open(file_name, "w")
            for objecto in frame_objects:
                for prop in objecto[1]:
                    imagem_numerada = cv2.putText(imagem_numerada,␣
↪str(objecto[0]), objecto[1]["centroide"], cv2.FONT_ITALIC, 1, (0,255,0),␣
↪thickness=2)
                    file.write("\n" + prop + ": " + str(objecto[1][prop]))
                file.write("\n")
            file.close()
```

```
            cv2.imwrite("imgs/propriedades/frame_" + str(frameCount) + ".jpg",␣
↪imagem_numerada)
            cv2.imwrite("imgs/classificacao/frame_" + str(frameCount) + ".jpg",␣
↪contoured_frame)

        frame = next_frame

        key = cv2.waitKey(int(1000/fps))

        if(key == 27 or frameCount == total_frames-1):
            cv2.destroyAllWindows()
            capture.release()
```

[16]:
```
path = "video/"
inVideo = "camera1.mov"

MotionDetection(path + inVideo)
print("ACABOU")
```

```
FPS: 25.0
Total Frames: 3065.0
ACABOU
```

Conclusão

Com este trabalho foi possível implementar um detetor de movimento com tracking de objectos baseado nos conhecimentos adquiridos na unidade curricular de Processamento de Imagem e Visão.

No decorrer do projeto foi possível obter e analisar propriedades necessárias para classificar a maioria dos objetos apresentados no vídeo fornecido, usando técnicas como subtração de fundo, conversão de imagens para níveis de cinzento, binarização de imagens, melhoramento de imagens, extração de componentes conexos, extração de propriedades e classificação de objectos.

A subtração de fundo permite-nos extrair a informação dos pixeis em movimento e atualizar o fundo de forma a que este seja resiliente a possível alterações luminosas ou instabilidade de câmera.

O melhoramento de imagem trata a imagem de forma a que a extração de componentes e propriedades contenham o mínimo de erros possíveis.

A extração de componentes conexos e propriedades determina os objetos presentes na imagem e trata individualmente todos estes objectos extraindo features que os caracterizam.

A classificação determina a categoria a que cada objecto pertende, indentificado-o. E o tracking de objectos permite que mantenhamos informação sobre os objectos que permanecem no plano de imagem.

Existem, no entanto, alguns apectos a serem melhorados, nomeadamente, zonas de convergência com outros objectos podem gerar uma classificação incorreta do objecto (exemplo: poste à frente do carro), zonas de próximidade entre objectos podem gerar uma classificação única quando deveriam ser classificados dois objectos distintos e zonas de limite da imagem, com a diminuição da área, podem fazer com que um carro, por momentos, seja identificado como sendo uma pessoa. Ainda

assim, após o término do trabalho laboratorial proposto concluo que o objectivo geral foi cumprido sem dificuldades de maior.

[ ]:

[ ]: