Licenciatura Engenharia Informática e Multimédia

Codificação de Sinais Multimédia

Trabalho 4

Docente:

Eng.º José Nascimento

Grupo: 10

Turma: 41D

Miguel Távora Nº45102

João Cunha Nº45412

Arman Freitas Nº45414

Data de entrega: 9/06/2019

Índice

Índice de Figuras

1. Introdução e Objetivos

O último trabalho prático surge como um seguimento da norma JPEG, sendo esta também aplicável na compressão de vídeo. O vídeo é uma sequência de imagens(*frames*) que vão sendo exibidas conforme a passagem do tempo.

Geralmente são exibidas em taxas entre as 24 e as 60 FPS, do *inglês frames per second*, ou seja, quadros por segundo.

Para a compressão de vídeo não é suficiente utilizar apenas o algoritmo de compressão JPEG. Isto pois o ficheiro continuaria a ocupar muito espaço no disco.

Desta forma é feita a compressão através da redução do Bit rate, da codificação intra-frame e da codificação inter-frame.

1.1. Objetivo

O principal objetivo deste ultimo trabalho prático, é implementar em Python algoritmos de compressão de vídeo, neste caso do MPEG-4, utilizando os seus princípios de codificação e na redução do Bit rate. Obtendo desta forma vídeos de alta qualidade com uma elevada taxa de compressão.

Iremos utilizar, as compressão inter-frame e intra-frame como mencionado anteriormente.

2. Desenvolvimento

2.1. Intra-frame

Inicialmente, são consideradas as imagens(*frames*) como intra-frames e é aplicada a norma JPEG a todas estas.

Intra-frame são geralmente são geralmente associadas a mudanças de cena ou objetos com oclusões. A codificação de intra-frames surge para fazer a remoção da redundância espacial, ou seja, são retiradas de cada *frame* informação que não é visível ao ser humano, ou, que não causa muito impacto na *frame* original. Esta informação é econtrada matemáticamente ao encontrar blocos da *frame* com frequências muito altas.

Através deste método é possível reduzir substancialmente o espaço ocupado pelas *frames* utilizadas durante o vídeo.

2.2. Inter-frame

2.2.1. Diferenças entre frames

De seguida são consideradas todas as imagens como inter-frames(P) à exceção da primeira, isto é, todos os macro blocos de cada *frame* são do tipo(P). O objetivo é criar P-*frames* que são a diferença entre a *frame* a codificar e a *I-frame*(primeira *frame* do vídeo), sem compensação de movimento.

Para a codificação inter-frame tira-se por base o facto de que durante uma sequência de *frames*, uma *frame* é muito semelhante à *frame* anterior e posterior.

As diferenças entre inter-frames são geralmente devidas ao movimento de objetos ou da câmara pelo que podem ser codificadas pela diferença entre *frames*.

A sua descodificação é feita através da operação contrária, a soma com a *i-frame*.

De modo a não obtermos valores negativos com a subtração, somamos 127. Um exemplo de uma *frame* subtraída e a mesma descodificada é a seguinte:

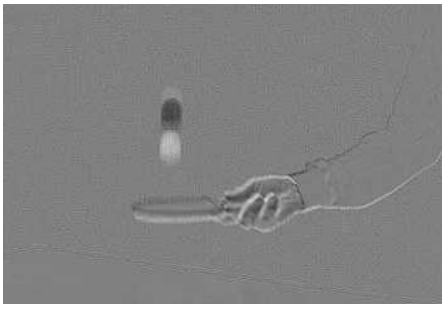

Figura 2 - *Frame* subtração

Figura 1 - *Frame* descodificada

2.2.2. Predição de *frame*

Finalmente mantendo também todas as imagens como inter-frames excepto a primeira, é necessário implementar a predição da *frame* a codificar, tendo por base a I-*frame*, fazendo compensação do movimento.

A *frame* que será transmitida é a diferença entre a *frame* a codificar e a sua predição.

2.2.2.1. Erro absoluto médio

Para a concretização desta parte foi primeiramente criada uma função que faz a medição do erro absoluto médio(MAE) entre dois blocos de tamanho (16x16).

A fórmula consta que:

$$(d_m, d_n) = \arg min \frac{1}{MN} \sum_{m=1}^{M}, \sum_{n=1}^{N} | x^t(m,n) - x^{t-1}(m - d_{m,n} - d_n)$$

2.2.2.2. Pesquisa

No seguimento foi necessário a implementação de uma função que realiza uma pesquisa. Neste sentido foi escolhida a *full-search*, que se insere nas pesquisas ótimas.

A pesquisa é feita ao bloco da *frame* a codificar numa janela de pesquisa de tamanho entre (-15 a +15) da I-frame.

Na remoção da redundância temporal para além da diferença entre frames temos ainda a estimação/compensação de movimento. Nesta parte e na última iremos abordar a estimação de movimento, ou seja, o vetor do movimento.

Esta metodologia possui algumas vantagens como: reduzir o bit rate, se a *frame* de estimação for boa o erro a transmitir é quase nulo e possui uma baixa entropia, logo a codificação é mais eficiente. Contudo tem algumas desvantagens como: introduzir atrasos e aumentar a complexidade computacional.

A estratégia de procura é avaliada segundo a eficiência da compressão do movimento, onde se divide a energia do bloco pela energia do resíduo. A complexidade computacional também é tida em conta, ou seja, o número de operações aritméticas efetuadas por cada bloco.

2.2.2.3. Frame de predição

Por fim é necessário percorrer os blocos da *frame* a codificar e construir a *frame* predita.

Na compensação de movimento é dividido cada frame em blocos não sobrepostos(macro-blocos). De seguida compara-se cada bloco com outros blocos da *frame* de referência dentro da janela de pesquisa feita anteriormente. Por fim determina-se qual o vetor do movimento ótimo.

Por fim, encontram-se alguns gráficos representativos das taxas de compressão/descompressão, SNR e tempo de compressão para todos os *frames* da sequência da bola:

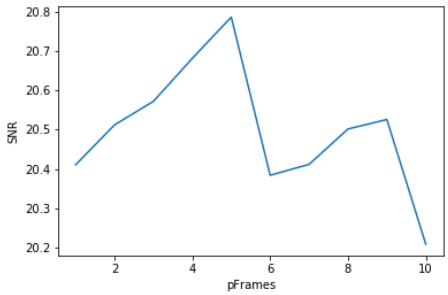

Figura 6 - SNR em função dos pFrames

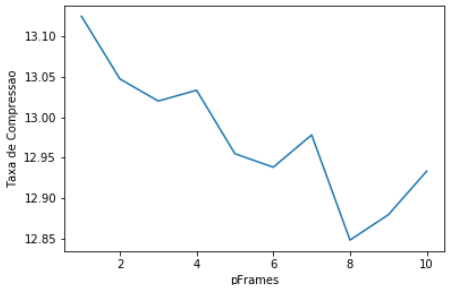

Figura 5 - Taxa de compressão em função dos pFrames

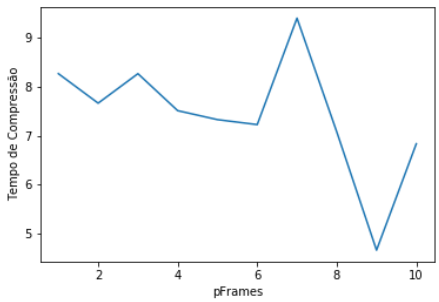

Figura 4 - Tempo de compressão em função dos pFrames

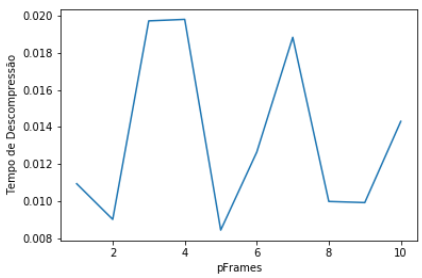

Figura 3 - Tempo de descompressão em função dos pFrames

3. Conclusões e Resultados

No presente trabalho o grupo realizou a norma de compressão de vídeo MPEG-4, do qual é possível obter uma elevada compressão com uma qualidade semelhante à qualidade do ficheiro inicial.

Conseguimos compreender os diferentes tipos de compressão de vídeos:

- Intra-frame
- Inter-frame
    - Diferença entre *frames*
    - Predição de *frame*

Numa plataforma de *streaming* de vídeos *online* percebemos que não se deve cometer o erro de apenas utilizar uma compressão intra-frames. Isto pois, não estamos apenas a transmitir uma imagem, mas sim várias *frames* por segundo. O vídeo é de facto, comprimido mas não tanto quanto a compressão inter-frame por predição de *frame* (calculando os vetores de movimento e a subtração de frames).

Durante a sua realização o grupo foi capaz de concretizar os objetivos estipulados inicialmente para o trabalho, contudo a implementação não foi a mais eficiente devido mais uma vez aos ciclos utilizados.

4. Anexos

4.1. Intra-frame

```
import cv2
from os import path
import numpy as np
from time import time
import matplotlib.pyplot as plt

def ex1():
    for i in range(1, 12):
        x_img = cv2.imread("../bola/bola_" + str(i) +".tiff",
cv2.IMREAD_GRAYSCALE)
        cv2.imwrite("bola_" + str(i) + ".jpeg", x_img,
(cv2.IMWRITE_JPEG_QUALITY, 50))

ex1()
```

4.2. Inter-frame (subtração de *frame*)

```
import numpy as np
import cv2


lenImagens = 11
imagens = []
for i in range (lenImagens):
   imagens.append(cv2.imread("../bola/bola_"+ str(i+1)+".tiff",
cv2.IMREAD_GRAYSCALE))

def codifica (arr):
    imagens = arr.copy()
    iFrame = None
    for i in range (len (imagens)):
        if i != 0:
            cv2.imwrite("codificado/pFrame_" + str(i) + ".jpeg", imagens[i],
(cv2.IMWRITE_JPEG_QUALITY,50))
            pFrame = cv2.imread("codificado/pFrame_" + str(i) + ".jpeg",
cv2.IMREAD_GRAYSCALE)

            pFrame = pFrame.astype('float64')
            final = pFrame - iFrame

            final += 128

            cv2.imwrite("codificado/pFrame_" + str(i) + ".jpeg", final)
        else:
            cv2.imwrite("codificado/pFrame_" + str(i) + ".jpeg", imagens[i],
(cv2.IMWRITE_JPEG_QUALITY,50))
            iFrame = cv2.imread("codificado/pFrame_" + str(i) + ".jpeg",
cv2.IMREAD_GRAYSCALE)
            iFrame = iFrame.astype('float64')



def descodificador_ex2():
    elFinal = []

    for i in range(11):

        elFinal.append(cv2.imread("codificado/pFrame_" + str(i) + ".jpeg",
cv2.IMREAD_GRAYSCALE))
```

```
        if i == 0:
            imagem_inicial = elFinal[i].astype('float64')
            cv2.imwrite("descodificado/pFrame_"+str(i)+ ".jpeg",
imagem_inicial)

        else:
            valor = np.array(elFinal[i], np.float64)

            valor -= 128
            valor += imagem_inicial



            valor = np.array(valor, np.uint8)
            cv2.imwrite("descodificado/pFrame_" + str(i) + ".jpeg", valor)

codifica (imagens)
descodificador_ex2()
```

4.3. Inter-frame (predição de *frame*)

4.3.1. Main

```
from tp4 import escreverDiferencas, lerDiferencas, predicao, escreverPredita

import cv2
from time import time
from os import path
import numpy as np
import matplotlib.pyplot as plt

def CalcularSNR (pr, original):
       return (10 * np.log10
(np.sum(np.sum(np.power(pr.astype('float'),2)))/np.sum(np.sum(np.power((pr.ast
ype('float')-original.astype('float')),2)))))

def ex3(obj):
    ex = None
    if obj == "bola":
        ex = ".tiff"
    else:
        ex = ".bmp"

    frames = []
    compressao = []
    descompressao = []
    taxasdeCompressao = []
    SNRs = []

    imagens = []
    for x in range(11):
       imagens.append(cv2.imread("../" + obj + "/" + obj + "_"+ str(x+1) + ex,
cv2.IMREAD_GRAYSCALE))

    iFrame = imagens.pop(0)

    cv2.imwrite("i_frame" + obj + ".jpeg", iFrame,
(cv2.IMWRITE_JPEG_QUALITY,50))
    iFrame = cv2.imread("i_frame" + obj + ".jpeg", cv2.IMREAD_GRAYSCALE)
```

```
    for i in range (len(imagens)):

        print ("######################" , (i + 1), "######################")
        frames.append(i + 1)

        time1 = time()
        frame_predita, vetor_mov = predicao(imagens[i], iFrame)
        cv2.imwrite("predicao/" + obj + "/predita" + str(i + 1) + ".jpeg",
frame_predita)

        file = "diferencas/" + obj + "/diferenca" + str(i) + ".jpeg"
        escreverDiferencas (imagens[i], frame_predita, file)
        tempo = time() - time1
        print ("Tempo de Compressão:", tempo)
        compressao.append(tempo)


        time2 = time()
        frame_dif = lerDiferencas(file)

        frame_p = escreverPredita(iFrame, frame_dif, vetor_mov)
        cv2.imwrite("descodificados/" + obj + "/p_frame" + str(i + 1) +
".jpeg", frame_p, (cv2.IMWRITE_JPEG_QUALITY,50))
        tempo = time() - time2
        print ("Tempo de Descompressão:", tempo)
        descompressao.append(tempo)

        original = path.getsize("../" + obj + "/" + obj + "_" + str(i + 2) +
ex)
        final = path.getsize("descodificados/" + obj + "/p_frame" + str(i + 1)
+ ".jpeg")
        taxa = original*1. / final*1.
        print ("Taxa de Compressão:", taxa)
        taxasdeCompressao.append(taxa)

        original = cv2.imread("../" + obj + "/" + obj + "_" + str(i + 2) + ex,
cv2.IMREAD_GRAYSCALE)
        final = cv2.imread("descodificados/" + obj + "/p_frame" + str(i + 1) +
".jpeg", cv2.IMREAD_GRAYSCALE)

        SNR = CalcularSNR(final, original)

        print ("SNR:", SNR)
        SNRs.append(SNR)

    plt.plot(frames, taxasdeCompressao)
    plt.xlabel("pFrames")
    plt.ylabel("Taxa de Compressao")
    plt.savefig("graficos/" + obj + "/Taxa de Compressão")
    plt.show()
    plt.close()

    plt.plot(frames, SNRs)
    plt.xlabel("pFrames")
    plt.ylabel("SNR")
    plt.savefig("graficos/" + obj + "/SNR")
    plt.show()
    plt.close()

    plt.plot(frames, compressao)
    plt.xlabel("pFrames")
    plt.ylabel("Tempo de Compressão")
    plt.savefig("graficos/" + obj + "/Tempo de Compressão")
    plt.show()
    plt.close()
```

```
    plt.plot(frames, descompressao)
    plt.xlabel("pFrames")
    plt.ylabel("Tempo de Descompressão")
    plt.savefig("graficos/" + obj + "/Tempo de descompressão")
    plt.show()
    plt.close()

ex3("bola")
```

4.3.2. Funções necessárias

```
import numpy as np
import cv2

def erroAbsolutoMedio(a, b):
    l = len(a)
    c = len(a[0])
    return np.sum(np.abs(a.astype('float64') - b.astype('float64'))) / (l*c)


def escreverDiferencas (p_frame, predita, filename):
    elFinal = p_frame.astype('float64') - predita.astype('float64')
    elFinal += 128
    cv2.imwrite(filename, elFinal, (cv2.IMWRITE_JPEG_QUALITY,50))

def lerDiferencas (filename):
    elFinal = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    elFinal -= 128
    return elFinal


def fullSearch (bloco, iFrame, pos):

    minimo = 9999999
    posicao = blocoFinal = None

    for i in range(15 * 2 + 1):
        i += pos[0]
        if i > len(iFrame) - len(bloco):
            continue

        for j in range(15 * 2 + 1):
            j += pos[1]

            if j > len(iFrame[0]) - len(bloco):
                break

            blocoI = iFrame[i:i+len(bloco), j:j+len(bloco)]

            erro = erroAbsolutoMedio(blocoI, bloco)

            if erro < minimo:
                minimo = erro
                posicao = (i, j)
                blocoFinal = blocoI

    return posicao, blocoFinal


def predicao(pFrame, iFrame):

    p_frame = np.zeros_like(pFrame)
    l = 0
```

```
    c = 0
    vetores = []

    for x in range (int(len(pFrame) / 16)):
        for y in range (int(len(pFrame[0]) / 16)):

            bloco16x16 = pFrame[l: l + 16, c: c + 16]
            nPos, blocoSem = fullSearch(bloco16x16, iFrame, (l, c))

            # Calcular cada vetor de movimento
            vetor = (nPos[0] - l, nPos[1] - c)
            p_frame[l: l + 16, c: c + 16] = blocoSem
            vetores.append(vetor)
            c += 16
        l += 16
        c = 0

    return p_frame, vetores



def escreverPredita(i_frame, diferenca, vetores):

    l = c = vIndex = 0

    predita = np.zeros_like(i_frame)

    for x in range(int(len(i_frame) / 16)):
        for y in range(int(len(i_frame[0]) / 16)):

            posX = vetores[vIndex][0] + l
            posY = vetores[vIndex][1] + c

            predita[l: l + 16, c: c + 16] = \
                                i_frame[posX: posX + 16, posY: posY + 16]
            vIndex += 1
            c += 16
        l += 16
        c = 0
    elFinal = predita + diferenca
    return elFinal
```