Universidade da Beira Interior
Departamento de Informática

TP - Multimédia: *Manipulação de Imagem, Áudio, Vídeo e Compressão*

Elaborado por:

Simão Andrade - a45464
Tiago Barreiros - a46118

Orientador:

Professor Doutor João Dias
Professor Bruno Degardin
Professor Vasco Lopes

Agradecimentos

Queremos agradecer a todas as identidades, que direta ou indiretamente, estiveram envolvidas neste trabalho desenvolvido no âmbito da Unidade Curricular (UC), Tecnologias Multimédia.

Como tal, começamos por agradecer ao nosso orientador Prof. Doutor João Dias, o nosso docente nas aulas teóricas e práticas, que suscitou e inseriu em nós, alunos, o interesse em saber mais, de nos tornar mais hábeis, proporcionando-nos todas as aprendizagens e métodos, necessários, para a realização deste projeto. No entanto, é importante realçar que este trabalho é também fruto da nossa expressiva investigação pessoal, visto que a linguagem abordada seria por nossa conta. De igual modo, realçar as aprendizagens obtidas no primeiro ano, como, por exemplo, a escrita de documentos em LaTeX, lecionada na UC - Laboratórios de Programação (LP), regida, no nosso ano, pelo Prof. Doutor Pedro M. Inácio, a quem devemos bastante reconhecimento e somos igualmente gratos.

Agradecemos também aos nossos colegas, e amigos, com o quais pudemos trocar ideias e um especial agradecimento às nossas famílias que estão neste momento a investir e a acreditar em nós. Da nossa parte um muito obrigado!

Resumo

Este projeto, consistiu em implementar um programa capaz de manipular e exercer operações sobre Imagem, Áudio, Vídeo e Compressão.

Este trabalho desenvolvido em `Python`, depressa começou a ganhar forma e expressão, contém vários *menus* bastante interativos, que proporcionam ao utilizador uma maior facilidade de compreensão e manuseamento do programa. Esta experiência, conta com 10 algoritmos de imagem, seis de Áudio, três de Compressão e três de vídeo. Contamos com o seu esquema representado na tabela resumo 1.

Operações sobre:	Algoritmos
Imagem	Operações Aritméticas: - Adição entre duas imagens; - Subtração entre duas imagens. Operações Lógicas: - `AND` Lógico entre duas imagens; - `OR` Lógico entre duas imagens; - `XOR` Lógico entre duas imagens. Aplicação de Filtros: - Transformação para Preto e Branco; - Transformação para Negativo; - Transformação no Inverso (horizontalmente); - Transformação no Inverso (verticalmente). outros - Comprimir imagem.
Áudio	Operações de Edição: - Cortar; - Adição; - Junção. Aplicação de Filtros: - Normalização; - Abrandar; - *Delay*.
Vídeo	Conversão de Vídeo: - Transformação para Negativo; - Transformação no Inverso.
Compressão	Técnicas de Supressão: - Compressão em *Lempel-Ziv-Welch* (LZW); - Técnica *Run-Length Encoding* (RLE); - Técnica de supressão de zeros ou espaços.

Tabela 1: Tabela Resumo dos algoritmos implementados.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Palavras-Chave

Áudio, Compressão, Imagem, Multimédia, Python.

Acrónimos

EI Engenharia Informática

LP Laboratórios de Programação

LZW *Lempel-Ziv-Welch*

MA Matemática Aplicada

RLE *Run-Length Encoding*

UBI Universidade da Beira Interior

UC Unidade Curricular

Capítulo

1

Introdução

1.1 Âmbito e Enquadramento

Encontra-se a analisar um relatório, elaborado por alunos do 2.º ano da Licenciatura Engenharia Informática (EI) da Universidade da Beira Interior (UBI).

Foi confecionado no contexto da Unidade Curricular (UC) Tecnologias Multimédia da UBI, desenvolvido maioritariamente à distância, visa familiarizar os alunos com os diferentes tipos de informação multimédia e como esta pode ser modificada e comprimida.

Abrange três áreas muito importantes do conhecimento, EI, Multimédia e Matemática Aplicada (MA), pois, desenvolvemos todo um programa que, para além de implementar algoritmos, exerce todo um cálculo matemático que na ausência dessas operações lógicas, seria impossível desenvolver este trabalho.

É importante abordar estes conceitos, entender como o analógico pode ser transformado no digital, como podemos tratar os dados e criar algo exato, trivial e essencial para o ser humano.

1.2 Motivação

Como referido anteriormente, este projeto une Programação com Matemática, que por sua vez torna possível a manipulação de tecnologias multimédia. Permite aos utilizadores usufruir de um misto de funcionalidades essenciais no nosso dia a dia e em diferentes contextos, quer por questões estéticas, redes sociais ou todo o tipo de desenvolvimento.

Nos dias de hoje, a manipulação de Imagens e Áudio é algo necessário, fundamental, pois vivemos na época do digital e a tecnologia promete.

Além disso, este trabalho foi realizado no âmbito das metas definidas pelo Prof. Doutor João Dias, como gerente da UC - Tecnologias Multimédia, como tal consideramos essa a nossa principal motivação.

1.3 Objetivos

Este trabalho pretende analisar, planear e implementar várias funções, de cariz matemático, desenvolvidas numa linguagem de programação à nossa escolha, bem como a sua criação e execução em ambiente de linha de comandos.

O principal objetivo do projeto, consiste em criar um programa de *software* e implementar funções, que permitam gerir operações multimédia, manipular imagens, áudios e compreender as diferentes etapas envolvidas no desenvolvimento do projeto, que visam:

1. **Imagem** - operações aritméticas, lógicas e aplicação de filtros;
2. **Áudio** - operações de edição e aplicação de filtros;
3. **Vídeo** - conversão de um vídeo a cores num vídeo a preto e branco, no seu negativo e inverso;
4. **Compressão** - compressão em *Lempel-Ziv-Welch* (LZW), técnica *Run-Length Encoding* (RLE), técnicas de supressão de zeros e espaços.

Além do desenvolvimento de *software*, como objetivos secundários vitais para a nossa vida académica, este trabalho permite uma maior aptidão na pesquisa e elaboração de projetos de uma forma colaborativa, promove o trabalho em equipa, de forma a adquirir métodos, experiência e um maior à vontade.

1.4 Abordagem

Começamos por abordar o problema, entender o que temos de fazer, quanto tempo levará a sua realização e o quão preparados temos de estar para a sua concretização.

A organização é o valor mais importante a defender, pois, na sua ausência impossibilitasse a criação de um método de trabalho funcional.

Com a experiência adquirida no primeiro ano, ambos decidimos trabalhar com o auxílio da plataforma *GitHub*, de forma a permitir-nos trabalhar à distância e acompanhar o desenvolvimento de cada um. Não foi necessário marcarmos datas, após definirmos os objetivos, fomos seguindo o ritmo de cada um e mantivemo-nos sempre em constante comunicação na plataforma *Discord*, que permitiu uma ágil comunicação, para facilitar todo o processo, pois o tempo é fundamental.

Quão mais antecipadamente terminarmos os objetivos, resolvermos os problemas, mais tempo teremos para rever detalhes, pequenas imperfeições e acrescentar melhorias. Esse tempo, precioso, pode abrir portas para um excelente trabalho.

Procedemos à sua realização, aprimoramento e inclusão de extras. O relatório foi o último ponto a ser concluído.

1.4.1 Distribuição de Tarefas

Ambos colaboramos para a realização do mesmo.
Na tabela 1.1, resumimos as tarefas elaboradas e a presença de cada um:

Tarefa	**Membro/s**
Imagem	Simão Andrade, Tiago Barreiros
Áudio	Simão Andrade, Tiago Barreiros
Vídeo	Simão Andrade
Compressão	Simão Andrade, Tiago Barreiros
Relatório	Tiago Barreiros

Tabela 1.1: Tabela segundo a participação mais notória de cada um.

1.5 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – refere o projeto que nos foi proposto, o seu âmbito e Enquadramento, a Motivação para a sua realização, os objetivos e a forma como foram abordados;

2. O segundo capítulo – **Algoritmos** – descrição do funcionamento dos algoritmos;

3. O terceiro capítulo – **Resultados** – apresenta os resultados de execução da aplicação, envolvendo comparações reais entre os algoritmos implementados;

4. O quarto capítulo – **Conclusões e Trabalho Futuro** – conclusão do projeto desenvolvido incluindo considerações sobre o desempenho e trabalho futuro por realizar.

Capítulo

2

Algoritmos

2.1 Introdução

Neste capítulo o objetivo é descrever as funções implementadas. O capítulo divide-se do seguinte modo:

- a secção 2.2, **Imagem**, explica dependências das funções de manipulação de Imagem do programa;

  Contém as suas características em três subsecções, subsecção 2.2.1 - **Operações Aritméticas**, subsecção 2.2.2 - **Operações Lógicas** e subsecção 2.2.3 **Aplicação de Filtros**.

- a secção 2.3, **Áudio**, explica as operações de áudio do programa;

  Contém as suas características em duas subsecções, subsecção 2.3.1 - **Operações de Edição** e subsecção 2.3.2 **Aplicação de Filtros**.

- a secção 2.4, **Vídeo**, explica as operações de conversão de vídeo do programa;

- a secção 2.5, **Compressão**, introduz uma breve descrição dos algoritmos de compressão do programa.

2.2 Imagem

2.2.1 Operações Aritméticas

1. Adição de duas Imagens.

- Começamos por inicializar uma imagem em branco;
- de seguida, usamos dois ciclos para percorrer a largura e a altura da imagem criada, enquanto obtemos e procedemos à adição dos pixeis das duas imagens a serem somadas, se essa soma for superior a 255, então o valor *standard* vai ser esse;
- por fim, colocamos o pixel na imagem criada e guardamos.

```
def adicao(primeira_imagem, segunda_imagem):

    # Inicializa a imagem em branco
    adicao_imagem = Image.new(primeira_imagem.mode,
        primeira_imagem.size, 'White')

    for i in range(0, adicao_imagem.width):
        for j in range(0, adicao_imagem.height):
            # Obtem o pixel das imagens originais
            pixel1 = primeira_imagem.getpixel((i, j))
            pixel2 = segunda_imagem.getpixel((i, j))

            R = pixel1[0] + pixel2[0]
            if R > 255:
                R = 255

            G = pixel1[1] + pixel2[1]
            if G > 255:
                G = 255

            B = pixel1[2] + pixel2[2]
            if B > 255:
                B = 255

            # Coloca o pixel na imagem
            adicao_imagem.putpixel((i, j), (R, G, B))

    # Guarda a imagem
    adicao_imagem.save("Imagens/adicao.jpg")
```

Excerto de Código 2.1: Adição de duas Imagens.

2. Subtração de duas Imagens.

- Começamos por inicializar uma imagem em branco;
- de seguida, procedemos ao método usado anteriormente, neste caso para a subtração, se esse cálculo for inferior a zero, então o seu valor será zero;
- por fim, colocamos o pixel na imagem criada e guardamos.

```
def subtracao(primeira_imagem, segunda_imagem):

    # Inicializa a imagem em branco
    subtracao_imagem = Image.new(primeira_imagem.mode,
        primeira_imagem.size, 'White')

    for i in range(0, subtracao_imagem.width):
        for j in range(0, subtracao_imagem.height):
            # Obtem o pixel das imagens originais
            pixel1 = primeira_imagem.getpixel((i, j))
            pixel2 = segunda_imagem.getpixel((i, j))

            R = pixel1[0] - pixel2[0]
            if R < 0:
                R = 0

            G = pixel1[1] - pixel2[1]
            if G < 0:
                G = 0

            B = pixel1[2] - pixel2[2]
            if B < 0:
                B = 0

            # Coloca o pixel na imagem
            subtracao_imagem.putpixel((i, j), (R, G, B))

    # Guarda a imagem
    subtracao_imagem.save("Imagens/subtracao.jpg")
```

Excerto de Código 2.2: Subtração de duas Imagens.

2.2.2 Operações Lógicas

1. AND lógico entre duas imagens.

    - Começamos por inicializar uma imagem em branco;
    - de seguida, usamos dois ciclos para percorrer a largura e a altura da imagem criada, obtemos os pixeis das duas imagens e aplicamos uma condição com a operação lógica pretendida;
    - por fim, colocamos o pixel na imagem criada e guardamos.

```
def and_imagem(primeira_imagem, segunda_imagem):

    # Inicializa a imagem em branco
```

```
    and_img = Image.new(primeira_imagem.mode, primeira_imagem.size
        , 'White')

    for i in range(0, and_img.width):
        for j in range(0, and_img.height):
            # Obtem o pixel das imagens originais
            pixel1 = primeira_imagem.getpixel((i, j))
            pixel2 = segunda_imagem.getpixel((i, j))

            pixel3 = tuple(pixel1 & pixel2 for pixel1, pixel2 in
                zip(pixel1, pixel2))

            # Coloca o pixel na imagem
            and_img.putpixel((i, j), pixel3)

    # Guarda a imagem
    and_img.save("Imagens/and.jpg")
```

Excerto de Código 2.3: `AND` lógico entre duas imagens.

2. `OR` lógico entre duas imagens.

   - Voltamos a inicializar uma imagem em branco;
   - usamos o método anterior e alteramos a condição com a operação lógica pretendida;
   - por fim, colocamos o pixel na imagem criada e guardamos.

```
def or_imagem(primeira_imagem, segunda_imagem):
    # Inicializa a imagem em branco
    or_img = Image.new(primeira_imagem.mode, primeira_imagem.size,
        'White')

    for i in range(0, or_img.width):
        for j in range(0, or_img.height):
            # Obtem o pixel das imagens originais
            pixel1 = primeira_imagem.getpixel((i, j))
            pixel2 = segunda_imagem.getpixel((i, j))

            pixel3 = tuple(pixel1 | pixel2 for pixel1, pixel2 in
                zip(pixel1, pixel2))

            # Coloca o pixel na imagem
            or_img.putpixel((i, j), pixel3)

    # Guarda a imagem
    or_img.save("Imagens/or.jpg")
```

Excerto de Código 2.4: `OR` lógico entre duas imagens.

3. XOR lógico entre duas imagens.

   - Começamos por inicializar uma imagem em branco;
   - procedemos ao processo idêntico e modificamos a condição com a operação lógica pretendida;
   - por fim, colocamos o pixel na imagem criada e guardamos.

```
def xor_imagem(primeira_imagem, segunda_imagem):

    # Inicializa a imagem em branco
    xor_img = Image.new(primeira_imagem.mode, primeira_imagem.size
        , 'White')

    for i in range(0, xor_img.width):
        for j in range(0, xor_img.height):
            # Obtem o pixel das imagens originais
            pixel1 = primeira_imagem.getpixel((i, j))
            pixel2 = segunda_imagem.getpixel((i, j))

            pixel3 = tuple(pixel1 ^ pixel2 for pixel1, pixel2 in
                zip(pixel1, pixel2))

            # Coloca o pixel na imagem
            xor_img.putpixel((i, j), pixel3)

    # Guarda a imagem
    xor_img.save("Imagens/xor.jpg")
```

Excerto de Código 2.5: XOR lógico entre duas imagens.

2.2.3 Aplicação de Filtros

1. Preto e Branco.

   - Começamos por inicializar uma imagem em branco;
   - de seguida, usamos dois ciclos para percorrer a largura e a altura da imagem criada, obtemos o pixel das duas imagens originais e criamos uma variável para inserir os valores do pixel convertidos para preto e branco;
   - por fim, colocamos o pixel na imagem criada e guardamos.

```
def preto_e_branco(imagem):

    # Inicializa a imagem em branco
    preto_branco_img = Image.new(imagem.mode, imagem.size, 'White'
        )
```

```
    for i in range(0, preto_branco_img.width):
        for j in range(0, preto_branco_img.height):
            # Obtem o pixel da imagem original
            pixel1 = imagem.getpixel((i, j))

            R = pixel1[0]
            G = pixel1[1]
            B = pixel1[2]

            bw = int((R + B + G) / 3)

            # Coloca o pixel na imagem
            preto_branco_img.putpixel((i, j), (bw, bw, bw))

    # Guarda a imagem
    preto_branco_img.save("Imagens/pretoebranco.jpg")
```

Excerto de Código 2.6: Transformação de uma imagem a cores numa em preto e branco.

2. Negativo.

   - Voltamos a inicializar uma imagem em branco;
   - usamos o método anterior, mas desta vez guardamos os três valores do pixel subtraindo esse valor a 255;
   - por fim, colocamos o pixel na imagem criada e guardamos.

```
def negativo(imagem):
    # Inicializa a imagem em branco
    negativo_img = Image.new(imagem.mode, imagem.size, 'White')

    for i in range(0, negativo_img.width):
        for j in range(0, negativo_img.height):
            # Obtem o pixel da imagem original
            pixel1 = imagem.getpixel((i, j))

            R = 255 - pixel1[0]
            G = 255 - pixel1[1]
            B = 255 - pixel1[2]

            # Coloca o pixel na imagem
            negativo_img.putpixel((i, j), (R, G, B))

    # Guarda a imagem
    negativo_img.save("Imagens/negativo.jpg")
```

Excerto de Código 2.7: Transformação de uma imagem no seu negativo.

3. Inverso Horizontal.

    - Começamos por inicializar uma imagem em branco;
    - de seguida, usamos dois ciclos para percorrer a largura e a altura da imagem criada, obtemos o pixel da imagem original e criamos uma variável para inserir os valores do pixel referente à diferença entre a largura dessa imagem, da imagem invertida e um;
    - por fim, colocamos o pixel na imagem criada e guardamos.

```
def invertida_horizontal(imagem):

    # Inicializa a imagem em branco
    invertida_img = Image.new(imagem.mode, imagem.size, 'White')

    for i in range(0, invertida_img.width):
        for j in range(0, invertida_img.height):
            # Obtem o pixel da imagem original
            pixel1 = imagem.getpixel((i, j))
            ip = (imagem.width - i) - 1

            # Coloca o pixel na imagem
            invertida_img.putpixel((ip, j), pixel1)

    # Guarda a imagem
    invertida_img.save("Imagens/invertida_horizontal.jpg")
```

Excerto de Código 2.8: Transformação de uma imagem no seu inverso(horizontalmente).

Inverso Vertical.

- Inicializamos uma imagem em branco;
- procedemos ao processo anterior, desta vez para os valores da altura;
- por fim, colocamos o pixel na imagem criada e guardamos.

```
def invertida_vertical(imagem):

    # Inicializa a imagem em branco
    invertida_img = Image.new(imagem.mode, imagem.size, 'White')

    for i in range(0, invertida_img.width):
        for j in range(0, invertida_img.height):
            # Obtem o pixel da imagem original
            pixel1 = imagem.getpixel((i, j))
            jp = (imagem.height - j) - 1
```

```
            # Coloca o pixel na imagem
            invertida_img.putpixel((i, jp), pixel1)

    # Guarda a imagem
    invertida_img.save("Imagens/invertida_vertical.jpg")
```

Excerto de Código 2.9: Transformação de uma imagem no seu inverso(verticalmente).

2.3 Áudio

2.3.1 Operações de Edição

1. Cortar parte de um Áudio.
    - Começamos por converter os tempos inseridos pelo utilizador para segundos;
    - de seguida, declaramos um novo áudio com o intervalo de valores inseridos pelo utilizador;
    - por fim, exportamos esse áudio.

```
def corta_audio(primeiro_tempo, segundo_tempo, audio):

    # Conversao dos tempos dados para segundos
    t1 = primeiro_tempo * 1000
    t2 = segundo_tempo * 1000

    # Declaracao do "audio_cortado" como sendo igual uma fracao do
        vetor do audio original e exportacao do mesmo
    audio_cortado = audio[t1:t2]
    audio_cortado.export('Som/corte.wav', format="wav")
```

Excerto de Código 2.10: Cortar parte de um Áudio.

2. Adiciona dois Áudios.
    - Declaramos o áudio como sendo igual à soma dos vetores dos áudios originais;
    - por fim, exportamos o mesmo.

```
def adiciona_audio(primeiro_audio, segundo_audio):
    # Declaracao do "add_audio" como sendo igual a soma dos
        vetores dos audios originais e exportacao do mesmo
    add_audio = primeiro_audio + segundo_audio
```

```
    add_audio.export('Som/adicao.wav', format="wav")
```

Excerto de Código 2.11: Adicação de dois Áudios.

3. Juntar dois Áudios.

   - Declaramos o áudio como sendo igual ao produto dos vetores dos áudios originais;
   - por fim, exportamos o mesmo.

```
def junta_audio(primeiro_audio, segundo_audio):
    # Declaracao do "juncao_audio" como sendo igual ao produto dos
        vetores dos audios originais e exportacao do mesmo
    juncao_audio = primeiro_audio * segundo_audio
    juncao_audio.export('Som/juncao.wav', format="wav")
```

Excerto de Código 2.12: Junção de dois Áudios.

2.3.2 Aplicação de Filtros

1. Normalizar um Áudio.

   - Definimos a variável "amplitude"como a diferença do valor da normalização pretendida pelo utilizador e o valor da amplitude do áudio;
   - de seguida, declaramos um novo áudio como a soma do original com a variável "amplitude";
   - por fim, exportamos o próprio.

```
def normaliza_audio(valor_dbfs, audio):
    amplitude = valor_dbfs - audio.dBFS

    # Declaracao do "audio_normalizado" sendo igual ao audio
        original + o valor da amplitude e exportacao do mesmo
    audio_normalizado = audio + amplitude
    audio_normalizado.export("Som/normalizado.wav", format="wav")
```

Excerto de Código 2.13: Normalização de um Áudio.

2. Abrandar um Áudio.

   - Declaramos uma variável que será o resultado da diferença de um com a percentagem de abrandamento indicada pelo utilizador;
   - depois multiplicamos o *frame rate* pelo resultado anterior;
   - por fim, exportamos o áudio.

```
def abranda_audio(velocidade, audio):

    audio_slowed = audio
    slowed = 1 - velocidade

    audio_slowed.frame_rate *= slowed
    audio_slowed.export("Som/slowed.wav", format="wav")
```

Excerto de Código 2.14: Abrandar um Áudio.

3. Colocar *Delay* num Áudio.

    - Criamos um áudio vazio baseado na quantidade de atraso pretendida pelo utilizador;
    - depois adicionamos esse áudio ao início do áudio original;
    - por fim, procedemos à sua exportação.

```
def atraso_audio(atraso_ms, audio):
    # Criacao de um audio vazio dado a quantidade de atraso pedida
    vazio = atraso_ms * 1000
    audio_delay = AudioSegment.silent(duration=vazio)

    # Adicao desse mesmo audio vazio ao inicio do audio original e
        exportacao do mesmo
    audio_delay += audio
    audio_delay.export("Som/delay.wav", format="wav")
```

Excerto de Código 2.15: *Delay* num Áudio.

2.4 Vídeo

1. Transformar um vídeo no seu Preto e Branco.

    - Começamos por obter a resolução do vídeo original e definir o seu formato;
    - a seguir, inicializamos o novo vídeo, procedemos à leitura dos *frames* e à conversão a preto e branco.

```
def pretoebrancovid(vid):
    # Determina a resolucao do video capturado
    largura_quadro = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    altura_quadro = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    tamanho = (largura_quadro, altura_quadro)

    # Define o formato do video
```

```
    fourcc = cv2.VideoWriter_fourcc('M', 'P', '4', 'V')
    fps = vid.get(cv2.CAP_PROP_FPS)

    # Inicializa o novo video
    novo_video = cv2.VideoWriter('Video/blacknwhite_video.mp4',
        fourcc, fps, tamanho)

    while True:

        # Leitura dos frames
        existeQuadro, quadro = video.read()

        if existeQuadro:
            for x in range(0, largura_quadro - 1):
                for y in range(0, altura_quadro - 1):
                    r, g, b = quadro[y, x]
                    cinza = r * 0.2125 + g * 0.7174 + b * 0.0721
                    quadro[y, x] = [cinza, cinza, cinza]
            novo_video.write(quadro)
        else:
            break
    novo_video.release()
```

Excerto de Código 2.16: Tranformação de um vídeo no seu Preto e Branco.

2. Transformar um vídeo no seu Negativo.

   - Começamos por obter a resolução do vídeo original e definir o seu formato;
   - a seguir, inicializamos o novo vídeo, procedemos à leitura dos *frames* e à conversão para o seu negativo.

```
def negativovid(vid):
    # Determina a resolucao do video capturado
    largura_quadro = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    altura_quadro = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    tamanho = (largura_quadro, altura_quadro)

    # Define o formato do video
    fourcc = cv2.VideoWriter_fourcc('M', 'P', '4', 'V')
    fps = vid.get(cv2.CAP_PROP_FPS)

    # Inicializa o novo video
    novo_video = cv2.VideoWriter('Video/negativo_video.mp4',
        fourcc, fps, tamanho)

    while True:
```

```
        # Leitura dos frames
        existeQuadro, quadro = video.read()

        if existeQuadro:
            for x in range(0, largura_quadro - 1):
                for y in range(0, altura_quadro - 1):
                    r, g, b = quadro[y, x]
                    quadro[y, x] = [255 - r, 255 - g, 255 - b]
            novo_video.write(quadro)
        else:
            break
    novo_video.release()
```

Excerto de Código 2.17: Tranformação de um vídeo no seu Negativo.

3. Transformar um vídeo no seu Inverso.

    - Executamos o mesmo processo anterior, neste caso para o seu inverso.

```
def invertidovid(vid):
    # Determina a resolucao do video capturado
    largura_quadro = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    altura_quadro = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    tamanho = (largura_quadro, altura_quadro)

    # Define o formato do video
    fourcc = cv2.VideoWriter_fourcc('M', 'P', '4', 'V')
    fps = vid.get(cv2.CAP_PROP_FPS)

    # Inicializa o novo video
    novo_video = cv2.VideoWriter('Video/invertido_video.mp4',
        fourcc, fps, tamanho)

    while True:

        # Leitura dos frames
        existeQuadro, quadro = video.read()

        if existeQuadro:

            # Considerar o novo quadro como uma copia do original
            novoquadro = quadro.copy()

            for x in range(0, largura_quadro - 1):
                for y in range(0, altura_quadro - 1):
                    novoquadro[y, (largura_quadro - x) - 1] =
                        quadro[y, x]
            novo_video.write(novoquadro)
```

```
        else:
            break
    novo_video.release()
```

Excerto de Código 2.18: Tranformação de um vídeo no seu Inverso.

2.5 Compressão

1. Compressão em LZW.

    - Definimos um dicionário com o conjunto de caracteres existentes numa mensagem;
    - Declaramos `P = primeiro simbolo` e `C = próximo simbolo`;
    - Se existir, `P + C`, então `P = P + C`, senão adiciona `P + C` ao dicionário;
    - Termina quando não há mais caracteres na mensagem para ler.

```
def lzw_compressao(string):
    dicionario = []

    for letra in string:
        if not dicionario.__contains__(letra):
            dicionario.append(letra)
    dicionario = sorted(dicionario)
    dicionario.remove('\x00')

    codigoP = ""

    P = string[0]

    for i in range(1, len(string)):
        C = string[i]
        if dicionario.__contains__(P + C):
            P = P + C
        else:
            codigoP = codigoP + str(dicionario.index(P) + 1)
            dicionario.append(P + C)
            P = C
    print("O racio de compressao e de: " + str(len(string) / len(
        codigoP)))
    return codigoP
```

Excerto de Código 2.19: Compressão em LZW.

2. Técnica RLE.

    - Conta o número de vezes que uma letra se repete e cria uma *flag*.

```
def rl_compressao(string):
    codigoRL = ""
    i = 0
    while i < len(string):

        conta = 1
        while i < len(string) - 1 and string[i] == string[i + 1]:
            conta += 1
            i += 1
        i += 1

        if conta >= 2:
            codigoRL += str(conta) + "!" + string[i - 1]
        else:
            codigoRL += string[i - 1]
    print("O racio de compressao e de: " + str(len(string) / len(
        codigoRL)))
    return codigoRL
```

Excerto de Código 2.20: Técnica RLE.

3. Técnica de supressão de zeros e espaços.

    - Praticamente igual ao anterior, mas invés de verificar se todos os elementos têm mais que duas repetições, verifica apenas os zeros e os espaços.

```
def zeros_compressao(string):
    codigoZeros = ""
    bits = 0
    i = 0
    while i < len(string):

        conta = 0
        while i < len(string) - 1 and (string[i] == "0" or string[
            i] == " "):
            conta += 1
            i += 1

        if conta >= 2:
            codigoZeros += "!" + str(conta)
            bits += 1
        else:
            i += 1
            codigoZeros += string[i - 1]
```

```
    print("O racio de compressao e de: " + str(len(string) / (len(
        codigoZeros) - bits)))
    return codigoZeros
```

Excerto de Código 2.21: Técnica de supressão de zeros e espaços.

2.6 Conclusão

Neste capítulo, pretendia-se que fosse feita uma abordagem às funções e fossem enunciadas algumas das características a que recorremos na sua elaboração.

Capítulo

3

Resultados

3.1 Operações sobre Imagem

3.1.1 *Menu*

Representado na figura 3.1

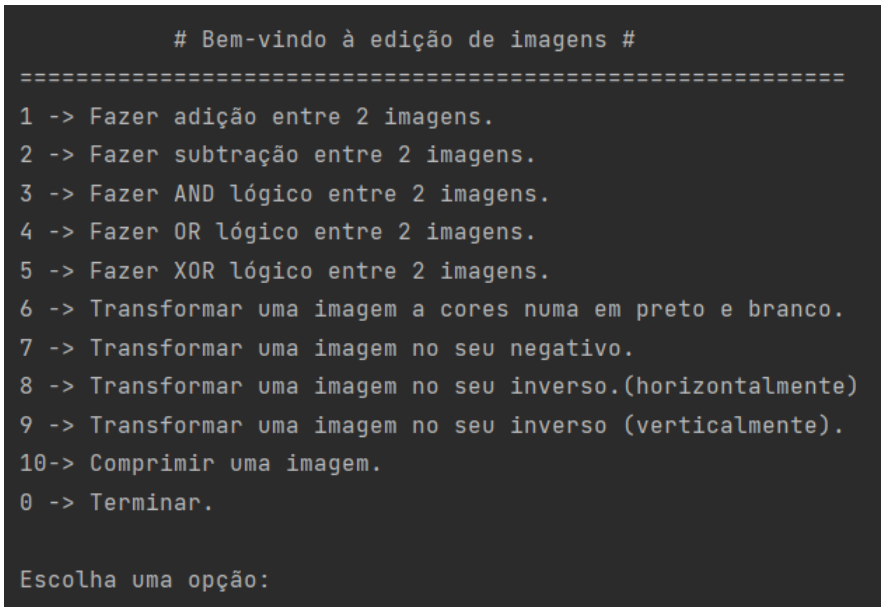

Figura 3.1: Imagem ilustrativa do *Menu* de Imagem.

3.1.2 Adição entre duas imagens

As imagens originais, bem como o resultado da sua adição, entram se representadas nas figuras 3.2 e 3.3.

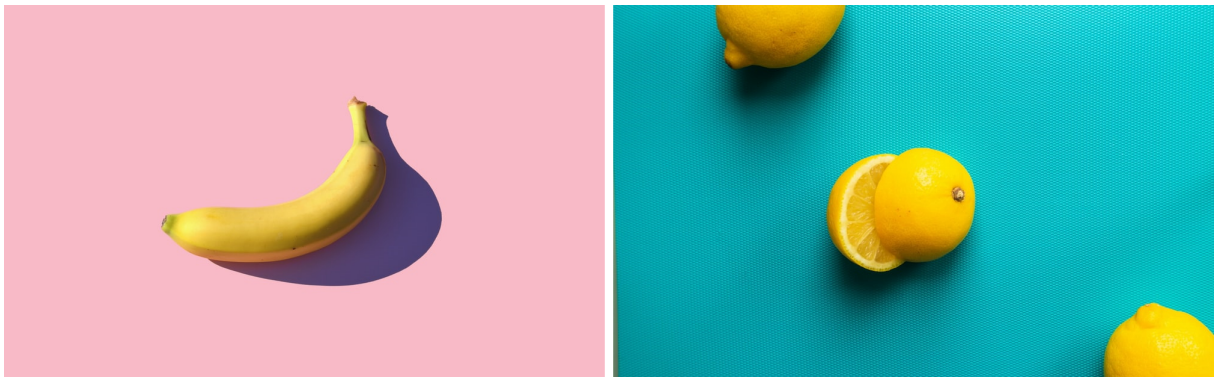

Figura 3.2: Representação das imagens originais.

Figura 3.3: Resultado da Adição de duas imagens.

3.1.3 Subtração entre duas imagens

Resultado da subtração de duas imagens, representado na figura 3.4

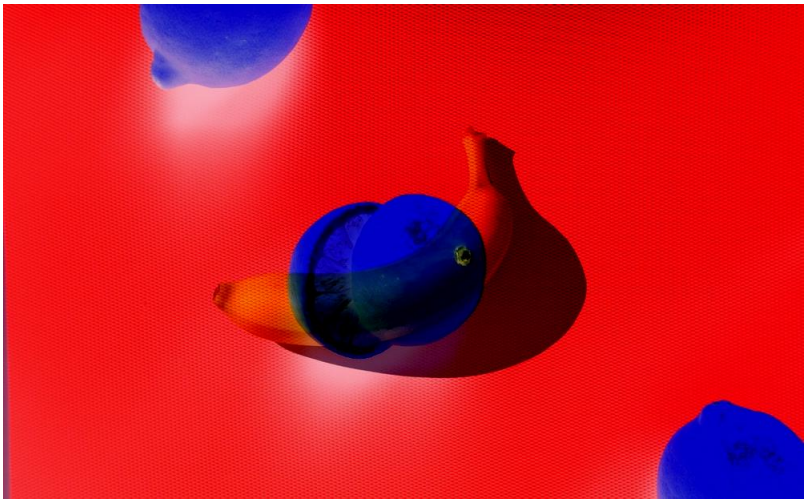

Figura 3.4: Resultado da Subtração de duas imagens.

3.1.4 `AND` Lógico entre duas imagens

Encontra-se representado na figura 3.5

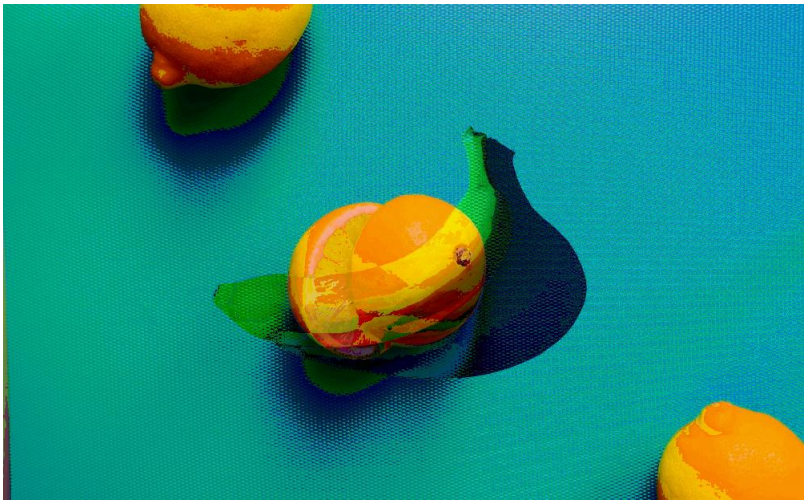

Figura 3.5: Resultado do `AND` Lógico entre duas imagens.

3.1.5 OR Lógico entre duas imagens

Encontra-se representado na figura 3.6

Figura 3.6: Resultado do OR Lógico entre duas imagens.

3.1.6 XOR Lógico entre duas imagens

Encontra-se representado na figura 3.7

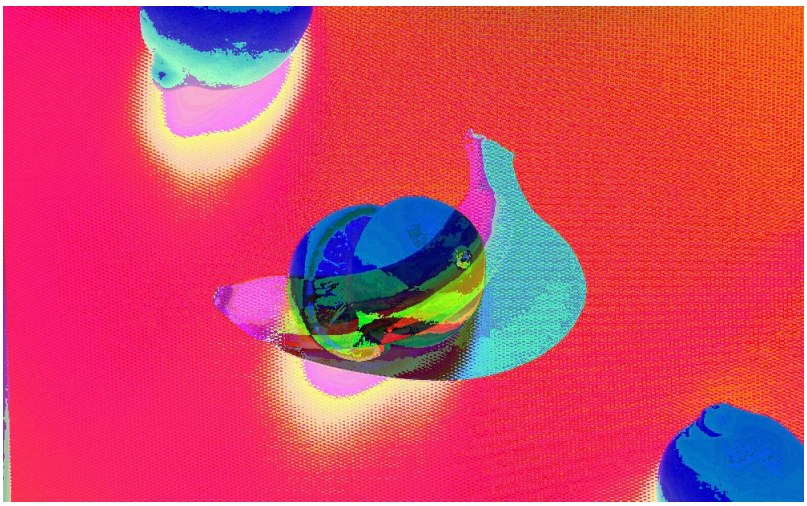

Figura 3.7: Resultado do XOR Lógico entre duas imagens.

3.1.7 Transformação Preto e Branco

Imagem original e transformada representadas, nas figuras 3.8 e 3.9, respetivamente.

Figura 3.8: Representação da imagem original.

Figura 3.9: Imagem ilustrativa da transformação a Preto e Branco.

3.1.8 Transformação no seu Negativo

Imagem original e transformada representadas, nas figuras 3.10 e 3.11, respetivamente.

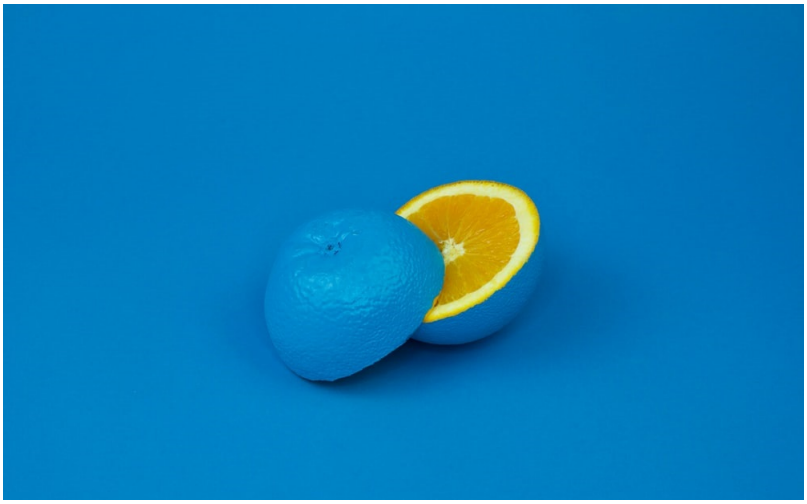

Figura 3.10: Representação da imagem original.

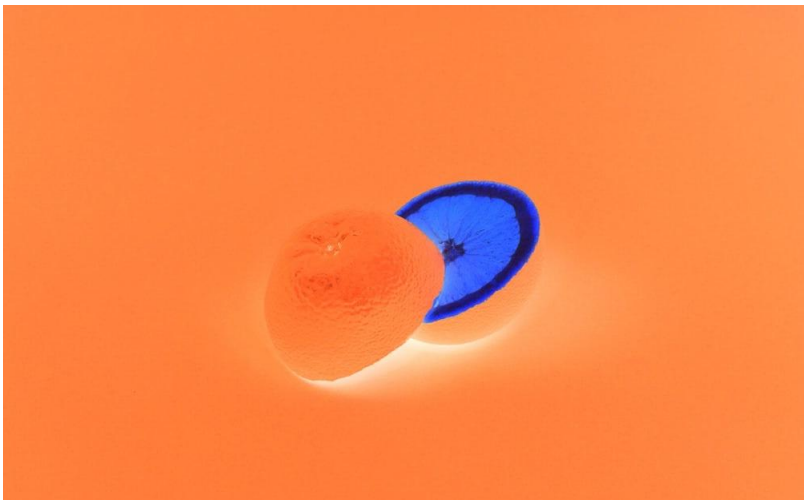

Figura 3.11: Imagem ilustrativa da transformação no seu Negativo.

3.1.9 Transformação no seu Inverso (horizontalmente)

Imagem original e o seu inverso, representados na figura 3.12, respetivamente.

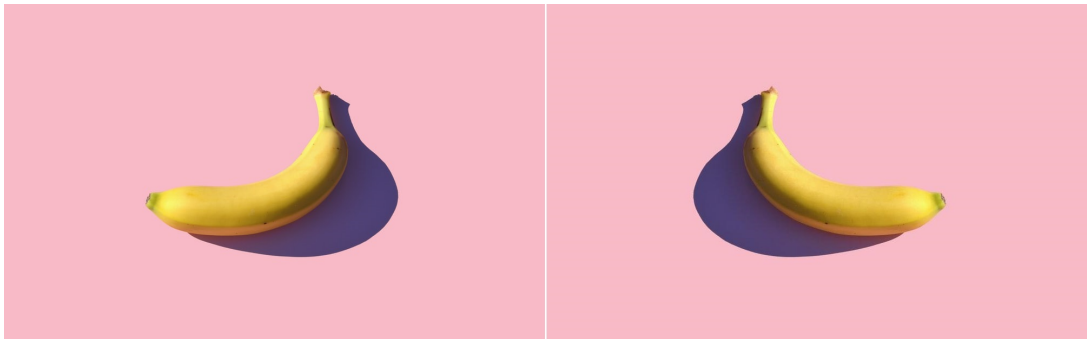

Figura 3.12: Imagem original e o seu Inverso (horizontalmente).

3.1.10 Transformação no seu Inverso (verticalmente)

Imagem original e o seu inverso, representados na figura 3.13, respetivamente.

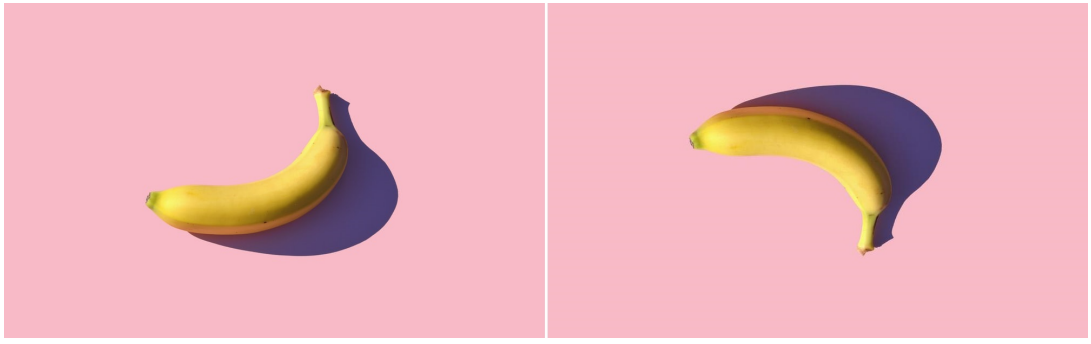

Figura 3.13: Imagem original e o seu Inverso (verticalmente).

3.1.11 Comprimir imagem

Representação da imagem original e da sua compressão de valor 20 e de valor dois, representados nas figuras 3.14 e 3.15

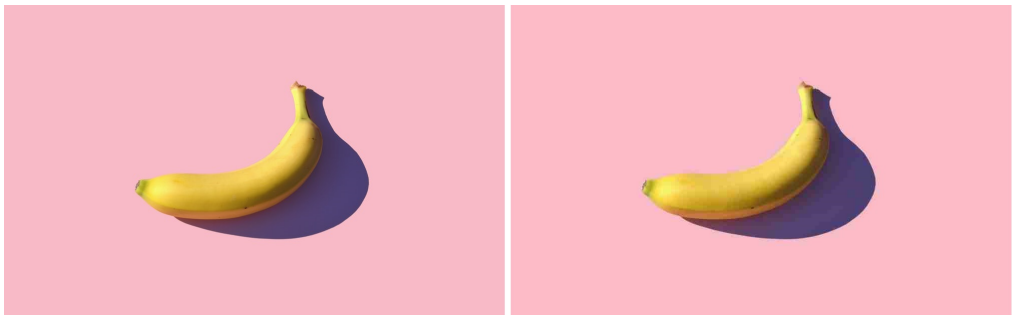

Figura 3.14: Imagem original e a sua compressão de valor 20.

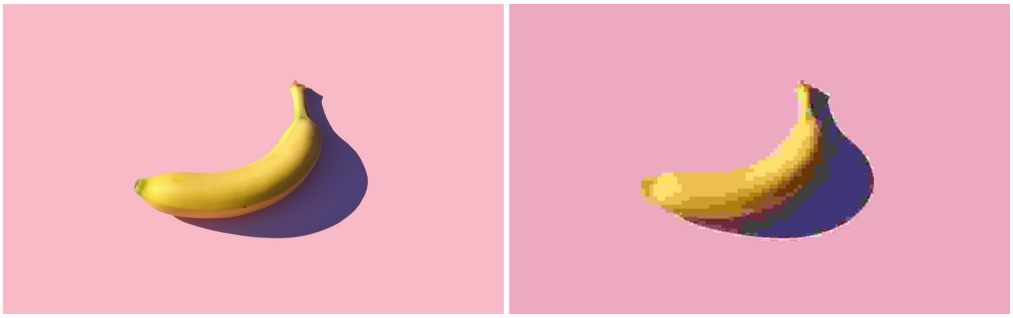

Figura 3.15: Imagem original e a sua compressão de valor dois.

3.2 Operações sobre Áudio

3.2.1 *Menu*

Representado na figura 3.16

```
          # Bem-vindo à edição de audio #
============================================================
1 -> Cortar parte de um audio.
2 -> Adiciona 2 audios.
3 -> Juntar 2 audios.
4 -> Normalizar um audio.
5 -> Abrandar um audio.
6 -> Colocar atraso (delay) num audio.
0 -> Terminar.

Escolha uma opção:
```

Figura 3.16: Imagem ilustrativa do *Menu* de Áudio.

3.2.2 Cortar

Imagem ilustrativa do corte de um áudio dos 16s aos 24s, bem como o seu resultado. Representado na figura 3.17.

Figura 3.17: Imagem ilustrativa do corte de um áudio dos 16s aos 24s.

3.2.3 Adição

Imagem ilustrativa da adição de dois áudios, bem como o seu resultado. Representado na figura 3.18.

Figura 3.18: Imagem ilustrativa da adição de dois áudios.

3.2.4 Junção

A função de Junção faz exatamente o que o nome indica, junta dois áudios.

3.2.5 Normalização

Imagem ilustrativa do resultado da normalização a -20dbs de um áudio. Representado na figura 3.19.

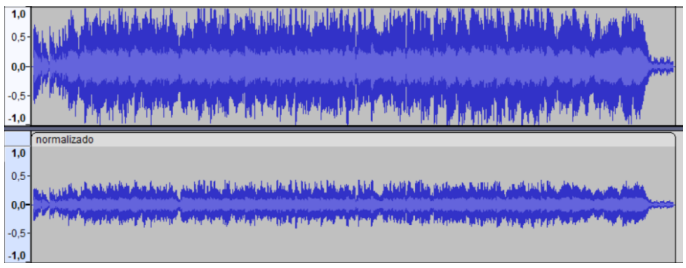

Figura 3.19: Imagem ilustrativa da normalização a -20dbs de um áudio.

3.2.6 Abrandar

Imagem ilustrativa do abrandamento de um áudio em 50%, bem como o seu resultado. Representado na figura 3.20.

Figura 3.20: Imagem ilustrativa abrandamento de um áudio em 50%.

3.2.7 *Delay*

Imagem ilustrativa do *delay* de 10s e 5s num áudio, bem como os seus resultados. Representados nas figuras 3.21 e 3.22.

Figura 3.21: Imagem ilustrativa do *delay* de 10s num áudio.

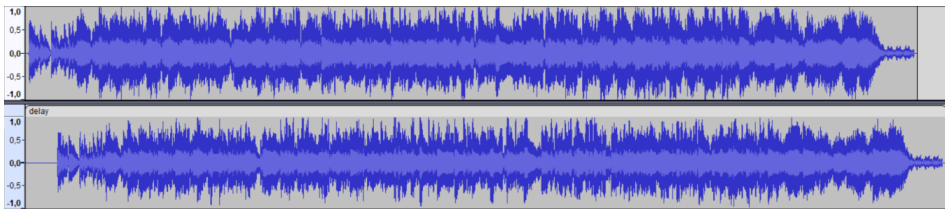

Figura 3.22: Imagem ilustrativa do *delay* de 5s num áudio.

3.3 Operações sobre Vídeo

3.3.1 *Menu*

Representado na figura 3.23

```
            # Bem-vindo à edição de video #
============================================================
1 -> Transformar um video no seu preto e branco.
2 -> Transformar um video no seu negativo.
3 -> Transformar um video no seu inverso.
0 -> Terminar.

Escolha uma opção:
```

Figura 3.23: Imagem ilustrativa do *Menu* de Vídeo.

3.3.2 Preto e Branco

Imagem ilustrativa do vídeo original e transformação representadas na figura 3.24, respetivamente.

Figura 3.24: Imagem ilustrativa do vídeo original e transformação a Preto e Branco.

3.3.3 Negativo

Imagem ilustrativa do vídeo original e transformação representadas na figura 3.25, respetivamente.

Figura 3.25: Imagem ilustrativa do vídeo original e transformação no seu Negativo.

3.3.4 Inverso

Imagem ilustrativa do vídeo original e transformação representadas na figura 3.26, respetivamente.

Figura 3.26: Imagem ilustrativa do vídeo original e transformação no seu Inverso.

3.4 Compressão

3.4.1 *Menu*

Representado na figura 3.27

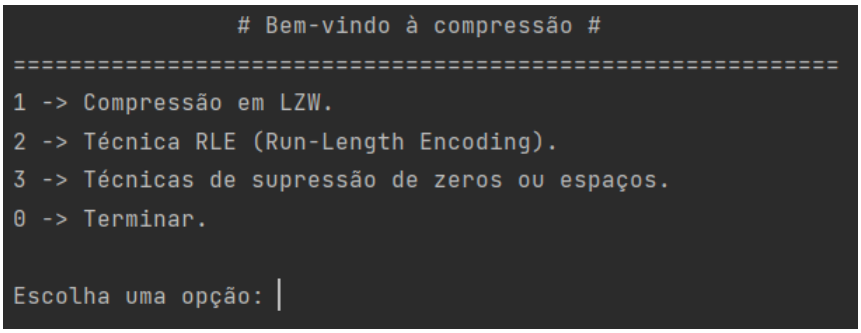

Figura 3.27: Imagem ilustrativa do *Menu* de Compressão.

3.4.2 LZW

Imagem ilustrativa, de um exemplo, da compressão em LZW. Representado na figura 3.28.

Figura 3.28: Imagem ilustrativa, de um exemplo, da compressão em LZW.

3.4.3 RLE

Imagem ilustrativa, de um exemplo, da técnica RLE. Representado na figura 3.29.

Figura 3.29: Imagem ilustrativa, de um exemplo, da técnica RLE.

3.4.4 Supressão de zeros e espaços

Imagem ilustrativa, de um exemplo, da técnica de supressão de zeros e espaços. Representado na figura 3.30.

Figura 3.30: Imagem ilustrativa, de um exemplo, da técnica de supressão de zeros e espaços.

3.5 Conclusão

Em suma, referimos todos os processos e resultados no nosso programa, bem como todas as características fornecidas no trabalho.

Capítulo

4

Conclusões e Trabalho Futuro

4.1 Conclusões Principais

Em conclusão, com este trabalho aprendemos bastante e pudemos colocar à prova todas as aprendizagens obtidas nesta UC. Ganhamos uma maior aptidão para trabalhar em equipa, elaborar relatórios técnicos, bem como a agilidade necessária e conceitos-chave para trabalhar com manipulação de tecnologias multimédia, dar uso à sua informação, programar e proceder à sua pesquisa.

Nada disto seria possível sem o conhecimento obtido na UC- Tecnologias Multimédia, as aulas mostraram-se objetivas e auto-contidas, o que permitiu criar uma ideia clara sobre o longo trabalho que tínhamos pela frente e como iríamos proceder à sua realização. Felizmente, conseguimos resolver com sucesso, todos os erros que captámos, produzimos um programa que faz exatamente o que é pedido, de forma eficaz, contida, evitando erros do utilizador e ajudando o mesmo com avisos, caso se encontre um pouco perdido.

Apresentámos, também, um relatório bem estruturado e devidamente "*linkado*". A escrita em LaTeX, permitiu-nos rever as competências adquiridas no primeiro ano, em torno da escrita e embelezamento de Relatórios Técnicos, definida como meta na UC- Laboratórios de Programação. Com base no trabalho, aprendemos bastante acerca da Manipulação de Imagem, Vídeo, Áudio e Compressão.

Tendo em conta os critérios de avaliação propostos, defendemos que foi uma justa proposta para os 8 valores pretendidos.

Por fim, este projeto também nos permitiu adquirir novas habilidades unipessoais que nos serão muito úteis no futuro.

4.2 Trabalho Futuro

Agradecemos, mais uma vez, ao Prof. Doutor João Alfredo Fazendeiro Fernandes Dias pela proposta, foi um projeto estimulante e desafiador, também trabalhoso.

O nosso objetivo, para já, foi cumprir as metas propostas, mas num possível trabalho futuro, seria interessante explorar uma componente gráfica, de forma a conseguir implementar um trabalho mais interativo e atraente.