Trabalho 4

23 de Dezembro de 2020

Nome: Christian Hideki Maekawa - **RA:** 231867

1 Introdução

O objetivo deste trabalho é implementar um algoritmo de esteganografia em imagens digitais.

2 O Programa

O programa foi implementado usando ubuntu 18.04.5 LTS e python 3.6.9. As bibliotecas utilizadas para este trabalho foram wget 3.2, click 7.1.2, numpy 1.18.5, matplotlib 3.2.2, opencv python 4.1.2.30. e pathlib 1.0.1.

A seguir coloquei os comandos para mostrar as configuração do ambiente.

[ ]:
```
!lsb_release -d
```

```
Description:    Ubuntu 18.04.5 LTS
```

[ ]:
```
!python --version
```

```
Python 3.6.9
```

2.1 Como executar

O programa foi desenvolvido para executar por linha de comando. O programa python chama `app.py` e contém 4 tipos de comandos:

- `download` : Cria uma ambiente de exemplo e baixa imagens.
- `codefile` : Comando para colocar uma mensagem dentro da imagem.
- `decodefile` : Comando que extrai a mensagem de dentro da imagem.
- `display-bits`: Essa função exibe uma imagem com apenas bit 0, 1, 2 e 7.

2.2 Entrada

Imagens baixadas utilizando o wget. As imagens foram baixados do link.

2.3 Saída

A saída do comando `download` são as imagens de exemplo do site e um arquivo tx. A saída do comando `codefile` é um arquivo com sufixo _coded que tem uma mensagem dentro da image. A saída do `decodefile` é a impressão da mensagem que está dentro da imagem. A saída do `display-bits` são 4 camadas de bits, 0 ,1, 2 e 7.

3 Parâmetros Utilizados

[ ]:
```
# Exemplos de comandos
python app.py download <str/path> # sem parâmetro criar uma pasta result
python app.py codefile -i <str/image> -mf <str/message_file> -c <int/channel> #␣
 ↪sem parâmetro usa o baboon.png
python app.py decodefile -i <str/image> -c <int/channel> # sem parâmetro usa o␣
 ↪baboon_coded.png
python app.py display-bits -i <str/image> # sem parâmetro usa o baboon.png
```

4 Solução

4.1 Download das imagens

As entradas das soluções são baixadas utilizando o wget. São baixadas 4 imagens de exemplo, baboon, monalisa, peppers e watch. Conforme a função abaixo. Essa função é utilizada pelo comando `download`.

[ ]:
```
def setup(path):
    """
    Essa função é responsável por baixar as imagens de exemplo e criar um␣
 ↪arquivo de txt.
    """
    [i.unlink() for i in path.rglob("*.txt")] # Procura por txt  e deleta
    [i.unlink() for i in path.rglob("*.png")] # Procura por imagens png e deleta
    wget.download("https://www.ic.unicamp.br/~helio/imagens_coloridas/baboon.
 ↪png",out=str(path))
    wget.download("https://www.ic.unicamp.br/~helio/imagens_coloridas/monalisa.
 ↪png",out=str(path))
    wget.download("https://www.ic.unicamp.br/~helio/imagens_coloridas/peppers.
 ↪png", out=str(path))
    wget.download("https://www.ic.unicamp.br/~helio/imagens_coloridas/watch.
 ↪png", out=str(path))
    f = open(path / "sample.txt", "a")
    f.write("This is a sample to make test!") # escreve a mensagem secreta
    f.close()
```

4.2 Bibliotecas utilizadas

Foi utilizado, click para fazer as configurações de comandos, wget para baixar as imagens, pathlib para fazer manipulação de arquivos e pastas, o matplotlib para fazer os gráficos, numpy para fazer as operações matemáticas de vetorização e opencv para carregar a imagem e salvar.

[ ]:
```
import click
import wget
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

4.3 Carregar imagem

Para carregar a imagen no programa é utilizada a função `loadImg`.É utilizada um dict para guardar a label da imagem e o conteúdo dela. Essa estrutura é utilizada para salvar imagem com nome da imagem.

[ ]:
```
def loadImg(path):
    """
    Essa função é responsável por carregar uma imagem. Path do pathlib é␣
 ↪utilizada para pegar uma imagem
    com final png da variável path o conteúdo vetorizado é armazenado no␣
 ↪dicionario de img.
    E o nome dessa imagem fica salvo no dict como referencia na hora salvar.
    """
    img = {}
    img[Path(path).stem] = cv2.cvtColor(cv2.imread(f"{path}"), cv2.
 ↪COLOR_BGR2RGB) # Carrega imagens BGR para RGB
    return img
```

4.4 Exibir os planos de bits 0 ,1 ,2 e 7

Para inspecionar a mudança no plano de bits é utilizada essa função para exibir 4 imagens com planos de bits diferente.

[ ]:
```
def applyMask(img,bit):
  """
  Função do primeiro trabalho responsável por retornar imagem com um plano de␣
 ↪bits
  """
  assert (bit>= 0) and (bit <=7), "Bit need to be between (0) and (7)"
  new_img = img&int(format(1 << bit,'08b'),2)
  return new_img*255

@clicode.command()
@click.option('file', '--image','-i',type=click.Path(),default=str(_output /␣
 ↪'baboon.png'))
```

```
def display_bits(file):
    """
    Essa função exibe uma imagem com apenas bit 0, 1, 2 e 7
    """
    bits = [0,1,2,7] # bits para inspecionar
    img = loadImg(file) # Carrega imagem
    label = list(img.keys())[0] # Pega o nome da imagem
    img = img[label] # Conteudo
    _shape = img.shape # Pega a dimensao da imagem
    imgs2 = []
    for i in bits:
        imgs2.append(applyMask(img.flatten(),i).reshape(_shape)) # Processa␣
↪imagem e armazena
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2) # Inicializa um espaço␣
↪para 4 imagens
    fig.suptitle(f"Imagens")
    fig.set_size_inches(15,15)
    for ax, img,b in zip(fig.get_axes(),imgs2,bits): # exibe imagens
        ax.imshow(img)
        ax.label_outer()
        ax.set_xticks([]), ax.set_yticks([])
        ax.set_title(f"{label} bit={b}")
    plt.show()
```

4.5 Chamada da função de inserção da mensagem na imagem

A função é responsável por pegar uma image e um arquivo txt e colocar a mensagem em um canal específico.

[ ]:
```
def phrasetobyte(_phrase):
    """
    Essa função é responsável por pegar uma variavel string  e converter cada␣
↪palavra
    em representação de byte e colocar numa lista.
    """
    _phrase = bytearray(_phrase,"utf8") # transforma string em vetor de byte
    _string = []
    for i in list(bytes(_phrase)): # Percorre cada letra
        _byte = []
        while (i>0): # Enquanto o vetor de byte do char nao zerar
            _byte.append(i%2) # Pegue o bit
            i = i//2 # Pegue bit seguinte
        while(len(_byte) < 8): # Preenche o restante do vetor com zero ate ter 8␣
↪bits
            _byte.append(0)
        _byte.reverse() # Inverte o vetor de bits para ter a representacao no␣
↪lado certo
```

```
        _string.append(_byte) # Guarda o vetor de byte na lista
    return _string # Returna uma lista de palavras com as letras em byte

# Comando para colocar uma mensagem dentro da imagem
@clicode.command()
@click.option('file', '--image','-i',type=click.Path(),default=str(_output /␣
 ↪'baboon.png'))
@click.option('msg', '--message_file','-mf',type=click.Path()␣
 ↪,default=str(_output / 'sample.txt'))
@click.option('channel','--channel','-c', type=click.IntRange(0, 2,␣
 ↪clamp=True),default=0)
def codefile(file,msg,channel):
    """Comando para colocar uma mensagem dentro da imagem"""
    img = loadImg(file) # carrega imagem
    f = open(msg, "r") # Abre o arquivo txt
    message = f.read() # le o conteudo
    f.close()
    custom_img = cv2.cvtColor(code(img[list(img.
 ↪keys())[0]],phrasetobyte(message),channel),cv2.COLOR_RGB2BGR) # Chama funcao␣
 ↪para codificar e da a ele lista de palavras em byte
    filename = Path(file).parent / f"{Path(file).stem}_coded.png" # nomeia o␣
 ↪arquivo que sera codificado
    print(filename) # exibi nome do arquivo
    if not cv2.imwrite(str(filename),custom_img): # Tenta salvar a imagem␣
 ↪codificada
        raise Exception("Could not write image")
```

4.6 Chamada da função de decoficação da mensagem da imagem

[ ]:
```
# Comando que extrai a mensagem de dentro da imagem
@clidecode.command()
@click.option('file', '--image','-i',type=click.Path(),default=str(_output /␣
 ↪'baboon_coded.png'))
@click.option('channel','--channel','-c', type=click.IntRange(0, 2,␣
 ↪clamp=True),default=0)
def decodefile(file, channel):
    """Comando que extrai a mensagem de dentro da imagem."""
    img = loadImg(file) # Carrega imagem codificada
    msg = decode(img[list(img.keys())[0]],channel)  # Chama funcao de␣
 ↪decodificacao e o canal
    print(msg) # Exibi a mensagem
```

5 Algoritmo

Essa parte do relatório estará focada em entender o código, mais a baixo estará a análise de cada método.

5.1 Codificação na imagem

[ ]:
```
def code(img,phrase,channel):
    """
    Essa função é responsável por pegar uma imagem RGB e a frase em byte e␣
↪passar para um canal da imagem.
    """
    _shape = img.shape # Pega a dimensão da imagem
    img = img.flatten() # Converte a imagem para uma dimensao
    _phrase = np.array(phrase).flatten() # Converte a frase para 1 dimensao
    for idx, v in enumerate(img): # percorre a imagem
        if channel < 3: # O canal precisa ser menor que 3 para ser inperceptivel␣
↪para visao humana
            img[idx] &= ~np.uint8(1<<(channel)) # Eu optei em zerar o conteudo␣
↪do canal isso deixa a imagem preta
            if  idx < len(_phrase) : # Enquanto houver mensagem para codificar␣
↪na imagem
                if  (_phrase[idx]): # verifica se o bit e 1 porque caso␣
↪contrario ja deixei zerado nao precisa fazer operacao de shift
                    img[idx] = v|np.uint8(_phrase[idx]<<channel) # Se for um␣
↪faco shift e faco um or para mudar o bit para 1
    return img.reshape(_shape).astype(np.uint8) # Retorna a imagem com tamanho␣
↪original e com uint8(valores devem ser entre ate 0-255)
```

5.2 Decodificação da mensagem na imagem

[ ]:
```
def decode(img,_channel):
    """
    Essa função é responsável por pegar uma imagem RGB e extrair a mensagem.
    """
    _string2 = [] # variavel onde ficarao os bits do caracter
    _size = img.size # tamanho da dimensao
    for idx, v in enumerate(img.flatten()): #Converte em um dimensao
        v &= np.uint8(1<<(_channel)) # Verifica o conteudo do bit do canal␣
↪escolhido
        if v > 0: # Se o bit estiver ativo ele guarda 1 caso contrario 0
            _string2.append(1)
        else:
            _string2.append(0)
    rec = []
```

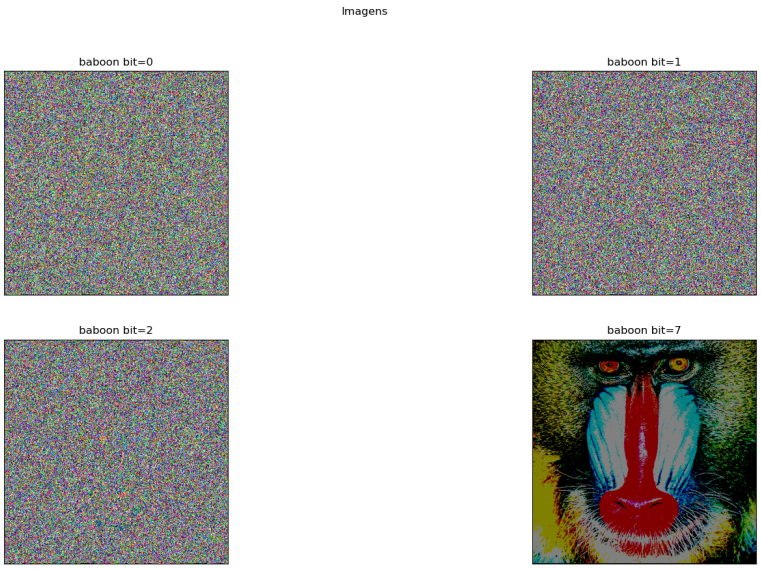
```
    for i in np.array(_string2).reshape(_size//8,8): # Converte o vetor de uma
↪dimensao no tamanho maximo de caracter suportado cada pixels com 8 bits
        rec.append(int(''.join(map(str, i)), 2))  #Concatena os bits
    msg = ''.join([chr(i) for i in rec]).replace(chr(0),"") # Conveter cada
↪vetor de 8 bits em caracter e concatena elas removendo o caracter 0
    return msg # retorna a mensagem
```

5.3 Análise de eficiência

Para esse experimento observou uma complexidade $N * M * 3$, tamanho da imagem com 3 canais. Ambas as soluções são rápidas.

6 Resultados

A seguir será apresentada um imagem com canal de bits 0, 1, 2 e 7.

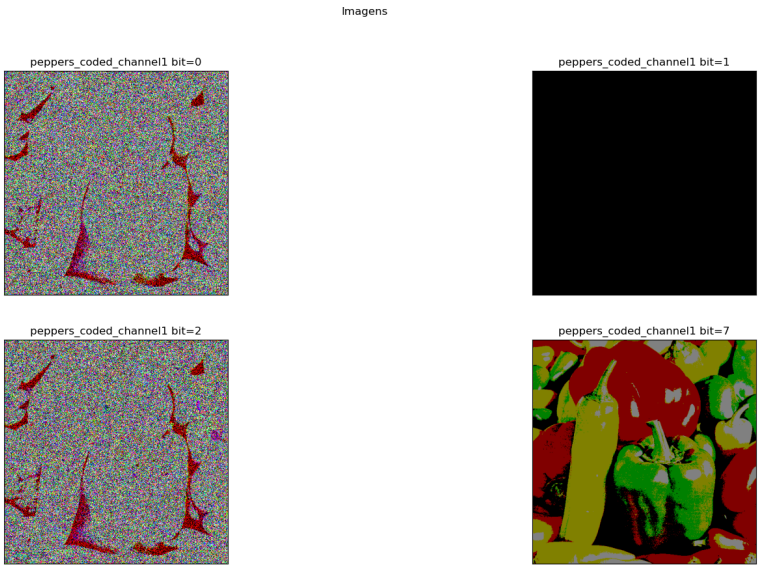


Ao aplicar a codificação no canal 0 é obtido o seguinte resultado. Como é feito um and no bit do canal então o valor do bit do canal é zerado tornando a imagem do canal totalmente preta.

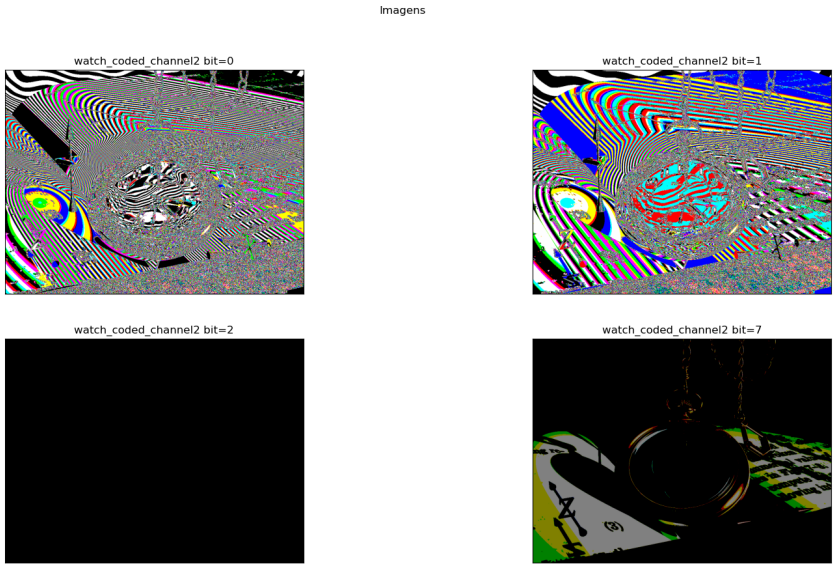
Mesmo trocando o canal 0 ou 1 ou 2 ainda obteve uma imagem que na minha percepção foram iguais a imagem original.

O mesmo aconteceu com as demais imagens.

Para o caso do peppers é possível notar que existe alguns traços da imagem da pimenta, mas mesmo utilizando o canal 0,1 e 2 não houve mudança visualmente perceptível.



O mesmo aconteceu com a imagem watch que aparentemente possuía informação relevante quando separado por canal de cada bit e que aparentavam fazer alguma diferença, mas no final não houve mudança visualmente perceptível nas imagens com mensagem dentro do canal, 0 ou 1 ou 2 com relação a imagem original elas visualmente pareciam iguais.



7 Conclusão

Para esse experimento obteve um resultado bastante interessante para conseguir atrelar alguma informação a imagem sem perceber visualmente um mudança. Pesquisando algumas aplicações por curiosidades encontrei sobre codificação de mensagens, para colocar alguma informação proprietária da imagem, também para conseguir colocar algum metadado a imagem como características da imagem para conseguir.