CC5508: Tarea 1

Gabriel De La Parra

Abril, 2017

Resumen

La presente tarea consiste en esconder texto dentro de una imagen de tal forma que tanto el texto como la imagen no se vean (afectada en el caso de la imagen) a simple vista. Dicha técnica se conoce como Esteganografía ([1]).

Para el caso de este trabajo, se trabajo con Python para generar un mensaje codificado, de tal forma que contenta el número de bits significativos de la imagen sobre los que se esconde el texto, el texto como tal y un mensaje de fin. Dicho mensaje de término se agregó para indicarle al código que la lectura debería llegar hasta dicho punto.

En el desarrollo de este trabajo se utilizaron operaciones de bits tales como máscaras, Shifts y operaciones binarias para esconder el mensaje codificado dentro de la imagen. El mismo procedimiento se utilizó para extraer el texto.

Introducción

La Esteganografía existe desde hace siglos. Según Wikipedia se aplicó en las primeras veces por Heródoto en el libro de las Historias. "*En este libro describe cómo un personaje tomó un cuadernillo de dos hojas o tablillas; rayó bien la cera que las cubría y en la madera misma grabó el mensaje y lo volvió a cubrir con cera regular. Otra historia, en el mismo libro, relata cómo otro personaje había rasurado a navaja la cabeza de su esclavo de mayor confianza, le tatuó el mensaje en el cuero cabelludo, esperó después a que le volviera a crecer el cabello y lo mandó al receptor del mensaje, con instrucciones de que le rasuraran la cabeza.*"

Para el caso de análisis de imágenes se puede ocupar para enviar y recibir mensajes ocultos en imágenes. Según

Desarrollo

La tarea está compuesta por dos partes. Codificación de un mensaje dentro de una imagen y decodificación de la imagen para obtener el texto.

Como se mencionó al inicio, el proceso para codificación y decodificación se realizaron por medio de operaciones a nivel de bits. El procedimiento se podría resumir de la siguiente manera:

1. Abrir la imagen y el texto
2. Calcular la máscara
3. Generar una lista de bits que contienen el texto codificado
4. Aplicar la codificacion a la imagen
5. Guardar la imagen

[1] https://es.wikipedia.org/wiki/Esteganografia

La apertura de la imagen se logra, como se vio en clases, a través de ScikitImage.IO. El texto se carga por OpenFile, con permisos unicamente de lectura.

```
image = io.imread(imageFilename)
text = open(textFilename,'r').read()
```

Para calcular la máscara, se calcula el valor máximo para un byte/pixel (0-255) y se le resta el $2^n bits$. De esta forma, por ejemplo, para un nbit de 3 se tiene una máscara de 0b 1111 1011.
Se presenta el código para su revisión:

```
def getMask(nbits):
    return 255-(2**(int(nbits)-1))
```

La generación de una lista con el texto codificado está compuesta de 3 partes:

- Agregar 4 bits que indican la cantidad de bits significativos con los que se encriptó la imagen.
- Convertir los caracteres (bytes) a bits.
- Agregar una marca para indicar el final del archivo.

Para agregar el número de bits significativos con los que se encriptó la imagen se desplaza hacia la derecha 1 uno la cantidad de veces que se especifica con nBits. El proceso se realiza con los operadores »¿z "&". El código se aprecia a continuación:

```
def getEncodedNbits(nbits):
    encodedNbits = []
    tmp = int(nbits)
    for i in range(0,4):
        encodedNbits.append(tmp & 1)
        tmp = tmp >> 1

    return encodedNbits
```

Convertir los caracteres a bits se logra de la misma manera. En la medida que se desplazan >> y se aislan &1se van agregando a la lista. El mismo proceso se ocupara para la marca del EOF. El código se puede apreciar a continuación.

```
    #Convertir cada caracter a bits y agregarlos a una lista:
    for char in text:
        byte = []
        tmp = ord(char)
        for i in range(0,8):
            byte.append(tmp & 1)
            tmp = tmp >> 1
        # Agregar byts a la lista:
        [encodedText.append(i) for i in byte[::-1]]
```

Para aplicar la codificación a la imagen se recorre la imagen como un arreglo de dos dimensiones, entendiendo que cada pixel se comporta como un byte, que compone una matriz que describe los distintos tonos de grises que componen la imagen. En este loop se aplica la máscara a cada byte y se agrega el texto desde la lista calculada anteriormente. El proceso se realiza por el siguiente código:

```
textCount = 0
    for row in range(0,len(image)):
        for column in range(0,len(image[row])):
            #TODO: Check que el texto entre en la imagen; Tirar exception;
            if(textCount < len(encodedText)):
                if(textCount < 4):
                    image[row][column] = (image[row][column] & getMask(1)) | (encodedText[textCour
                else:
                    image[row][column] = (image[row][column] & mask) | (encodedText[textCount] <<

                textCount += 1
```

Ha de notarse que los primeros 4 bits, que describen los nBits, se aplican siempre al bit menos significativo para que el decodificador los pueda leer desde la misma entrada. Como se revisará más adelante, la idea de agregarlos en el LSB es no alterer visualmente la imagen.

Para la decodificación, se realiza el proceso inverso. En primera instancia se leen los nbits de los primeros 4 bits y posteriormente se lee el resto de los caracteres a un arreglo. Se termina este proceso cuando se encuentra el EOF designado. El proceso se puede apreciar por el siguiente código:

```
for row in range(0,len(image)):
    for column in range(0,len(image[row])):
        if textCount < 4:
            encodedText.append(image[row][column] & 1)
        else:
            if textCount == 4:
                nbits = 8*encodedText[0] + 4*encodedText[1] + 2*encodedText[2]+ 1*encodedText[

            encodedText.append((image[row][column] >> (nbits-1)) & 1)
            if((textCount+5)%8 == 0 and textCount > 36):
                if(decodeText(encodedText[-32:]) == END_OF_FILE):
                    return decodeText(encodedText[4:-32])

        textCount += 1
```

Resultados

El resultado de aplicar la codificación tiene distintos resultados según el bit significativo que se ocupe. Como se sugiere en el enunciado del problema, es recomendable aplicarlo para un N¡=2. Para N mayores, la imagen empieza a distorcionarse por la modificación de los colores de la misma. En las siguientes imagenes se puede apreciar los distintos resultados:

Imagen 1. Nbits = 1.

Imagen 2. Nbits = 2.

Imagen 3. Nbits = 3.

Imagen 4. Nbits = 4.

Imagen 5. Nbits = 5.

Imagen 6. Nbits = 6.

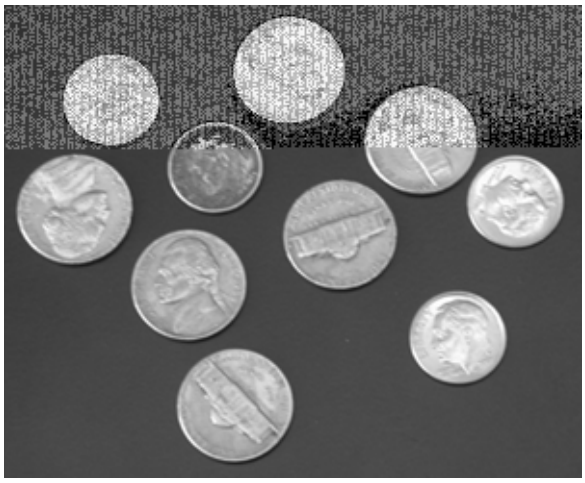

Imagen 7. Nbits = 7.

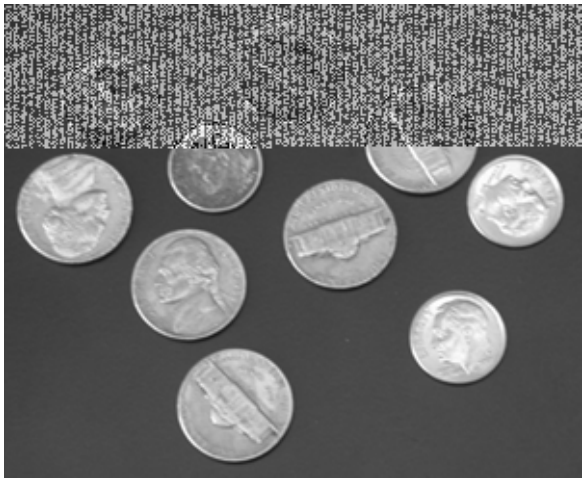

Imagen 8. Nbits = 8.

Conclusiones

El proceso de codificación de texto en imágenes permite esconder texto. Una de las limitaciones principales tiene que ver con que el texto que se utilice debe ser menor en longitud al tamaño de la imagen.

Lo anterior podría compensarse cuando se utilizan imánenes en colores ya que se triplica la cantidad de bytes disponibles para la codificación. Si se ocupan imágenes con transparencias *alpha*. Podría agregarse una dimensión más, siempre y cuando considerando que sería más oportuno ha-

cerlo con el bit menos significativo.

Uno de los principales problemas para manejar la encriptación tuvo que ver con el órden de los bits en la lista de los bits.

Otro punto interesante fue el delimitador final que se utilizó. Se considera que podria ser posible incluir el tamaño de la imagen al principio, sin embargo esto sería variable dependiento del tamaño.

Finalmente se puede mencionar que existen varios paquetes públicos para realizar esta encriptación. Algunos con varias opciones de configuración.