UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE INGENIERÍA Y CIENCIAS

Tarea 1

Gaspar Gumucio G.

Profesor: José M. Saavedra
Código: CC5508

Fecha de Entrega:
4 de octubre del 2020

1. Introducción

La esteganografía es un área que se dedica a estudiar las técnicas de encubrimiento de información. Se puede esconder cualquier tipo de información dentro de otra de tal forma que pase desapercibida y solo pueda ser revelada por alguien que sepa de antemano la manera de revelarla. Esto puede aplicarse para cualquier tipo de información, ya sea gráfica, de texto, audio, vídeos, etc. Un ejemplo cotidiano son los billetes, que al ser iluminados con luz ultravioleta revelan una marca distintiva que asegura que no son falsos.

Se sabe que la esteganografía se a utilizado desde tiempos lejanos, ya en el libro de las Historias de Herodoto, entre los años 484 y 430 a. C, se habla de esta técnica haciendo referencia a tablones con escrituras recubiertas de cera o mensajes tatuados en la cabeza de esclavos de confianza. También en la 2da Guerra Mundial fue muy utilizada para enviar mensajes sin que pudieran ser interceptados por los rivales.

Hoy en día se sigue ocupando, especialmente en seguridad. En plataformas como internet que pueden ser muy inseguras al momento de enviar información. Con esto por ejemplo se pueden marcar documentos con sellos de agua para proteger derechos de autor.

En el presente trabajo se tiene como objetivo ocultar una imagen dentro de otra. Tanto la imagen a ocultar como la que oculta pueden estar tanto en escala de grises como en rgb. Para lograr esto el procedimiento general será utilizar los bits menos significativos de la imagen portadora (la que oculta) y reemplazarlos por bits que contengan la información de la imagen a ocultar.

2. Diseño e Implementación

Para lograr el objetivo de ocultar la imagen implementamos un programa en python llamado *hide.py*. Este recibe como input dos imagenes donde la primera es la portadora y la segunda la que se ocultará. Un ejemplo en la terminal sería el siguiente:

```
$ python hide.py imagenA.jpg imagenB.png
```

El programa *hide.py* es el que esconde imagenB.png en imagenA.jpg y guarda la nueva imagen como imagenA_hide.png.

Luego se crea otro programa llamado *unhide.py* que reciba como input la imagen procesada anteriormente imagenA_hide.png y nos muestre la imagen oculta.

2.1. Hide.py

2.1.1. Lectura de input y guardado de imagen

Para leer el input que colocamos en la consola se importa el módulo sys. Luego utilizando skimage se leen las imagenes con la función imread que las entrega en el formato de matrices.

```
imagenA = pai_io.imread(imagenAfile)
imagenB = pai_io.imread(imagenBfile)
```

Teniendo la matriz ya procesada y con la información oculta solo queda usar la función imsave de skimage y colocarle un sufijo _hide.png al nombre del archivo.

```
x = imagenAfile.rsplit(".")
io.imsave(x[0]+'_hide.png',a)
```

2.1.2. Soporta Imagen

Previo a ocultar una imagen es necesario saber si es posible guardarla. Para esto se tiene que tener en cuenta el número de bits menos significativos que usaremos y el tamaño de las imágenes. Como cada píxel de la imagen es un número de 8 bits el espacio necesario es 8 veces el tamaño de la imagen a ocultar. Para guardar las dimensiones se ocuparon 33 bits adicionales debido a que el largo y ancho los representamos en números de 16 bits. El bit restante se utiliza para saber si la imagen es rgb (bit = 1) o grises (bit = 0).

La cantidad de bits que necesitamos para guardar debe compararse con la de bits disponibles que se obtiene de multiplicar la cantidad de bits menos significativos por el tamaño de la imagen A. En caso de no cumplirse esta condición mostramos un aviso de que la imagen no es soportada y no se guarda.

```
def soportaImagen(imagenA, imagenB, c):
    if 1 + 2*math.ceil(16/c) + imagenB.size*math.ceil(8/c) > imagenA.size:
        return False
    else:
        return True

def oculta(imagenA, imagenB, c):
    if not soportaImagen(imagenA,imagenB, c):
        print("Espacio insuficiente")
    else:
```

En el código la variable c es el número de bits menos significativos que utilizamos. Se ocupó la función cielo para ponderar el espacio que se gana al aumentar c.

2.1.3. Ocultar dimensiones

Para facilitar el trabajo convertimos las matrices a un arreglo unidimensional con reshape. Usando operadores de desplazamiento se eliminan de la imagen A sus bits menos significativos. Se crea un arreglo con la información de las dimensiones. Para guardar estos 2 números de 16 bits es necesario descomponer sus bits en $\lceil 16/c \rceil$ partes y cada una de esas partes se guardan en los primeros c bits menos significativos de los primeros pixeles de la imagen A. En la siguiente imagen hay una explicación simplificada para números de 4 bits:

imagenB	1101	1010		
imagenA	1111	1011	1010	1100
imagenA sin los 2 bits menos significativos	1100	1000	1000	1100
imagenA con B guardada	1101	1010	1011	1110

```
#guarda el valor del largo y ancho de la imagenB
a = np.reshape(imagenA, -1)
n = math.ceil(16/c)
dim = np.array([imagenB.shape[0],imagenB.shape[1]], dtype= np.uint16)
for i in range(n):
```

```
    a[(2*i+1):(2*i+1+2)] += dim - ((dim >> c) << c)
    dim = dim >> c
```

2.1.4. Ocultar pixeles

Para guardar los pixeles de la imagen el proceso es análogo. La diferencia es que ahora los bits de cada pixel se deben dividir en $\lceil 8/c \rceil$ por ser números de 8 bits.

```
#guarda la imagenB
b = np.reshape(imagenB, -1)
p = math.ceil(8/c)
for i in range(p):
    a[(1+2*n+b.size*i):(1+2*n+b.size*i+b.size)] += b - ((b >> c) << c)
    b = b >> c
```

2.1.5. Tipo rgb o gris

Para esta parte se terminó ocupando un píxel de la imagen A para guardar en su último bit un cero si era gris o 1 si era rgb. En esta parte puede que se pierdan algunos bits disponibles pero se consideró despreciable.

```
if len(imagenB.shape) == 2:
    a[0] += 0
else:
    a[0] += 1
```

Terminado lo anterior solo queda devolver la imagen a su tamaño original con reshape y guardarla.

2.2. Unhide.py

La función unhide debe extraer los datos guardados, para eso se hace un proceso inverso. Donde primero extraemos los bits menos significativos y luego recordando el orden en que fueron guardados se van sumando al mismo tiempo que usamos operadores desplazamiento para que los bits vayan en el lugar que estaba originalmente y se ponderen de forma correcta.

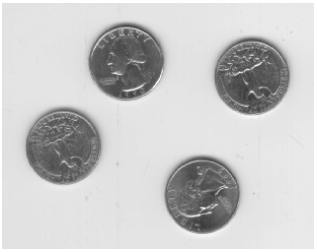
Además de esto hay que extraer y usar las dimensiones que fueron guardadas en los primeros pixeles para poder pasar de el array de una dimension a sus dimensiones originales.

```
def desoculta(imagenA,c):
    n = math.ceil(16/c)
    p = math.ceil(8/c)
    a = imagenA - ((imagenA >> c) << c)
    a = np.reshape(a, -1)
    b = np.zeros(2, dtype = np.uint16)
    d = a[1:1+2*n].astype(np.uint16)
    for i in range(n):
        b += d[2*i:2*i+2] << c*i
    alto = b[0]
    ancho = b[1]
    r = 0
    if a[0] == 0:
        r = 1
    else:
        r = 3
    tamaño = int(alto)*int(ancho)*r

    imagenB = np.zeros(tamaño, dtype = np.uint8)

    for i in range(p):
        imagenB += a[(1+2*n+tamaño*i):(1+2*n+tamaño*i+tamaño)] << c*i
    if a[0] == 0:
        imagenB = np.reshape(imagenB, (alto, ancho))
    else:
        imagenB = np.reshape(imagenB, (alto, ancho, 3))

    return imagenB



plt.imshow (desoculta(imagenA,c), cmap = 'gray')
plt.axis('off')
plt.show()
```

3. Resultados y Discusión

A continuación una imagen de 308x242 que se intenta guardar en una de 500x750:

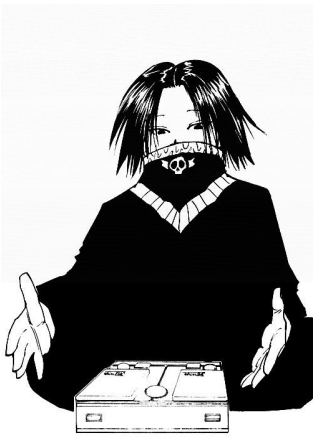
(a) Imagen A (500x750 pixeles)



(b) Imagen B (308x242 pixeles)

Como claramente $308*242*8 > 500*750$ no es posible guardarla solamente utilizando 1 bit menos significativo, pero con 2 bits menos significativos se puede ya que $308 * 242 * 8 < 500 * 750 * 2$. Luego de guardarla con 2 lsb (bits menos significativos) se fue aumentando la cantidad de lsb hasta el punto en que se considerara que la imagen ya no cumplía la función de ocultar debido a que se notaba intervenida.

(a) 2 bits



(b) 3 bits

(c) 4 bits

(d) 5 bits

Se puede ver por lo tanto que cuando se usan 5 bits menos significativos el fondo blanco de la imagen se vuelve claramente gris y en el sector superior de la imagen. Al ser este efecto ineludiblemente notorio se decide no utilizar más de 4 bits menos significativos.

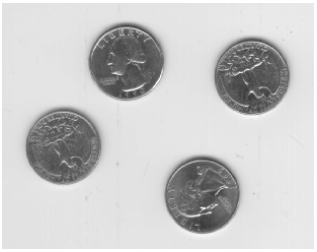
A continuación se muestra distintos ejemplos utilinzado 4 lsb.



(a) ImagenA (b) ImagenA procesada (c) ImagenB

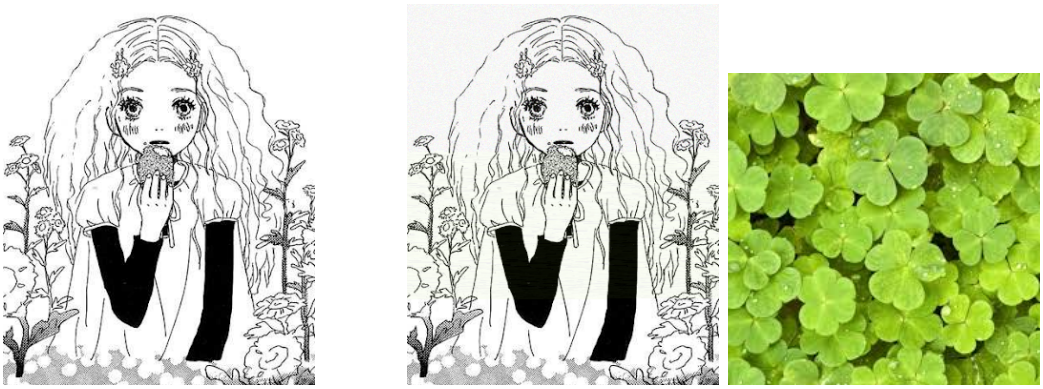
Figura 3: ImagenA(500x667x3) ImagenB(500x750)





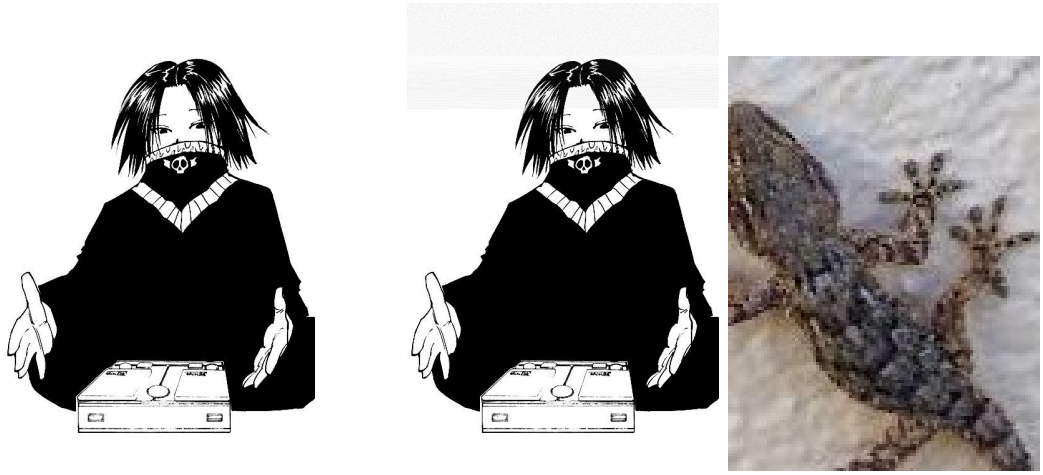
(a) ImagenA (b) ImagenA procesada (c) ImagenB

Figura 4: ImagenA(500x347x3) ImagenB(308x242)



(a) ImagenA (b) ImagenA procesada (c) ImagenB

Figura 5: ImagenA(327x400x3) ImagenB(225x225x3)



(a) ImagenA (b) ImagenA procesada (c) ImagenB

Figura 6: ImagenA(500x750) ImagenB(103x137x3)

4. Conclusiones

Analizando la forma en la que cambia la imagen A al utilizar sus bits menos significativos se concluye que no se deberían utilizar más de 4 bits menos significativos para ocultar información. Esto porque a partir del 5 bit la imágen se nota evidentemente intervenida, perdiendo su propósito.