Esteganografía en Imágenes

Joaquín Pérez Araya
Departamento de Ciencias de la Computación
Universidad de Chile
Santiago, Chile
joaquin.perez.a@ug.uchile.cl

Resumen—

Introducción

A lo largo de la historia de la humanidad ha existido la necesidad de envíar información ocultos, ya sea para transmitir información sensible, como Ejemplos... Uno de los modos es a través de la Esteganografía, que viene de ... En este documento se mencionará una implementación de Esteganografía sobre imagenes ... que consiste en ... la ... ejemplos de Esteganografía en imagenes (usar [1]) más específicamente un método bastante simple que consiste en... (tablita, algoritmo bonito, blablabla)

Dentro del texto actúal

Diseño e Implementación

Para la implementación del programa se utilizó Python 3.6 con las librerías `numpy`, `sci-kit-image` y `argparse`. El código está dividido en 4 partes: Utilidades, Entrada/Salida, Codificación y Decodificación.

Entrada y Salida

Para el uso externo del programa por medio de comandos se utilizó la librería `argparse` que viene por defecto en Python. Leer los comandos dados y traducirlos a llamados directos a las funciones de codificación o decodificación, tal como se pide en los requisitos de la tarea.

Utilidades

El módulo de utilidades (`util.py`) están las funciones auxiliares que utilizan la codificación y decodificación entre las de las cuales se destacan:

- `image_read`[1], `image_write` y `text_read`: Son las funciones para leer los archivos externos que se van a utilizar para el proceso de codificación y decodificación.
- `text_to_ascii` y `ascii_to_text`: La primera se utiliza para codificar el mensaje mientras que la segunda para decodificar. Se usan para transformar una cadena de texto a una lista de números donde cada caracter es un número de la lista y viceversa.
- `to_binary` y `to_int`: Se utilizan para codificar y decodificar, ya que reduce el problema de inserción de bits a modificar las cadenas de texto en binario[2] de los valores del canal de imagen.
- `join_binaries`: Se usa modificar la cadena de texto binaria para codificar un caracter en éste, une dos cadenas de texto, reemplazando los últmos n caracteres de la cadena por los n caracteres del segundo. Retorna un entero.
- `last_value`: Se utiliza para obtener los bits menos significativos de un número, para obtener el caracter que está codificado es éste.

Codificación

La codificación consta de una única función que dada la una dirección de imagen, una dirección texto y un número de bits n realice todo el proceso de abrir la imagen, obtener el primer canal de información para modificarlo con la finaldiad de agregarle la información que corresponde al texto.

Dado el texto a codificar, éste se convierte el texto a codificar a una lista de números[3] donde cada letra se le asigna un número que corresponde al valor en `ascii` del caracter, finalmente se agrega un último elemento 0 para agregar un final de codificación para el decodificador. Dado el primer canal de la imagen[4], se copia y en esta copia se codifica en los cuatros bits menos significativos del primer pixel $(0, 0)$ el número de bits que se van a usar en los siguientes para facilitar la acción de decodificación. Posteriormente, se utiliza la lista creada anteriormente para tomar los números de forma ordenada y traducirlos a binario, para insertarlos iterativamente.
Por cada cadena de texto binaria se toma sus últimos n bits y se insertan[5] hasta que la cadena de texto ya no se puede insertar complentamente, por lo que la ésta se inserta al final de la siguiente cadena binaria del siguiente carácter. Hasta agotar todos los caracteres de la lista.

Dado a que cada símbolo en `ascii` es un número de 8 bits y en cada píxel se elige la cantidad de bits a codificar, éste método permite codificar $\lfloor (p-1)(bits/8) \rfloor - 1$ caracteres como máximo dentro de una imagen con p píxeles en total,

[1]La función que está implementada en el código es la que está en `pai_io.py` dentro del repositorio del curso.

[2]Con cadena de texto binaria se refiere a una cadena de texto que es la representación binaria de un número.

[3]Usando `text_to_ascii`.

[4]Se usa el primero para evitar diferenciación al usar imagenes a color y en blanco y negro.

[5]Se insertan usando `join_binaries`

ya que en el primer pixel de la imagen siempre se va a utilizar para guardar la cantidad de bits utilizado en los 4 bits menos significativos de éste, tambien se necesita codificar un caracter adicional para indicar que no se debe seguir decodificando. De forma inversa dado c caracteres se necesitan $\lceil (c+1)(8/bits) \rceil + 1$ pixeles.

Decodificación

La decodficación sólo necesita la imagen para decodificar. Se extrae el primer canal de color, ya que durante la codificación se utilizó el primer pixel para codificar la cantidad de bits, se obtiene esta cantidad n para proceder a rearmar los caracteres tomando los valores de los primeros n bits en binario de los valores del canal de color hasta obtener 8 o más, para sacar los primeros 8 caracteres, transformalos en un número y agregarlos a la lista que se va a convertir en texto. El proceso se termina cuando se inserta en la lista el número 0 o en su forma binaria `00000000`.

Experimentación

Inicialmente para el testeo del funcionamiento inicial de la implementación se utilizó una imagen en negro de `10x10` pixeles con la finalidad de verificar vía simple inspección la codificación/decodificación.

Figura 1. El texto `Hola`, codificado usando 2, 4 y 8 bits para almacenar los datos (de izquierda a derecha).

Es mucho más notable el rojo con una codificación con más bits que con menos, sin embargo el coste de codificar con menos bits es que se necesita una imagen más grande para poder agregar un mismo mensaje. Usando la fórmula obtenida anteriormente en el diseño de la codificación, obteniendo un gráfico que es exponencialmente decreciente respecto a los bits.

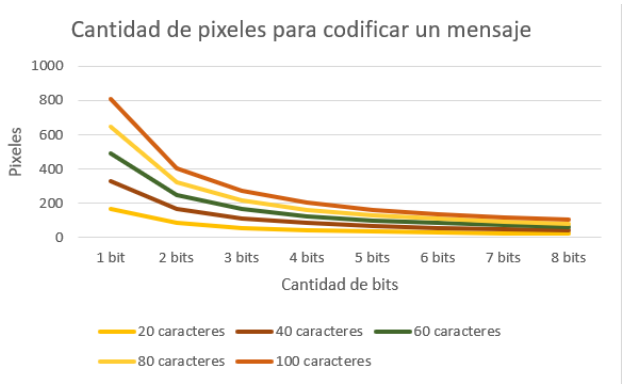

Figura 2. Gráfico que compara la cantidad de pixeles necesarios para codificar un mensaje con bits y cantidad de caracteres fijas.

Conclusión

Muy bien me gustó, póngame un 7.0 tkm.

Referencias

[1] Cheddad, A., Condell, J., Curran, K., & Mc Kevitt, P. (2010). Digital image steganography: Survey and analysis of current methods. Signal Processing, 90(3), 727–752.