Steganography

SHIVANI B : 1BI20CS160
ULLAS B R : 1BI20CS182
VAITHEESWARAN J : 1BI20CS183
YATISH S : 1BI20CS194

Agenda

Dem

- Introduction
- A brief look into various techniques
- Code Snippets
- Demonstration
- Applications
- Conclusion and Future Work

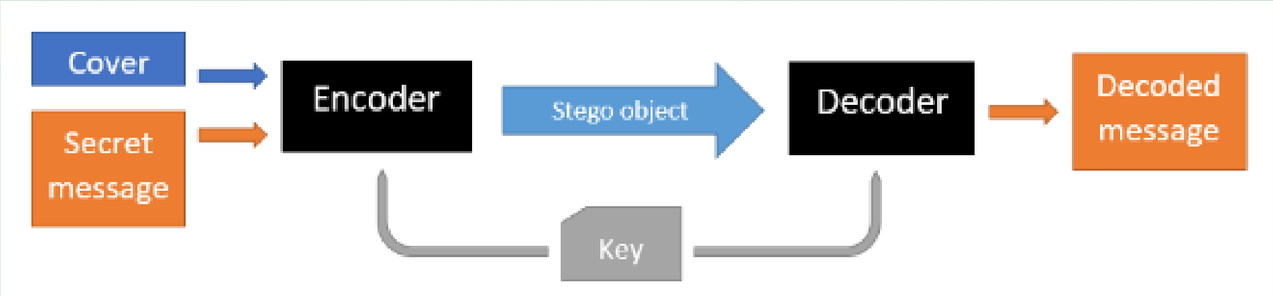
Introduction

- Steganography is the practice of concealing information within other forms of data or media, such as images, audio files, or text, in order to hide its existence.
- The goal is to hide the presence of the hidden information, making it difficult for unauthorized individuals to detect or access.
- These techniques involve embedding the hidden data into the carrier file in a way that doesn't significantly alter the appearance or characteristics of the original file.

- To achieve this, an innocuous-looking cover object is chosen. Then, to accommodate the secret message, the original, also called the cover,is slightly modified by the embedding algorithm.

- This embedding procedure must result in very little deviation from the original.

- The receiving party can, with the prior shared knowledge of where to look, decode the hidden embedded message from the stego object. Should it be intercepted, an eavesdropper would be oblivious to any secret communication taking place.



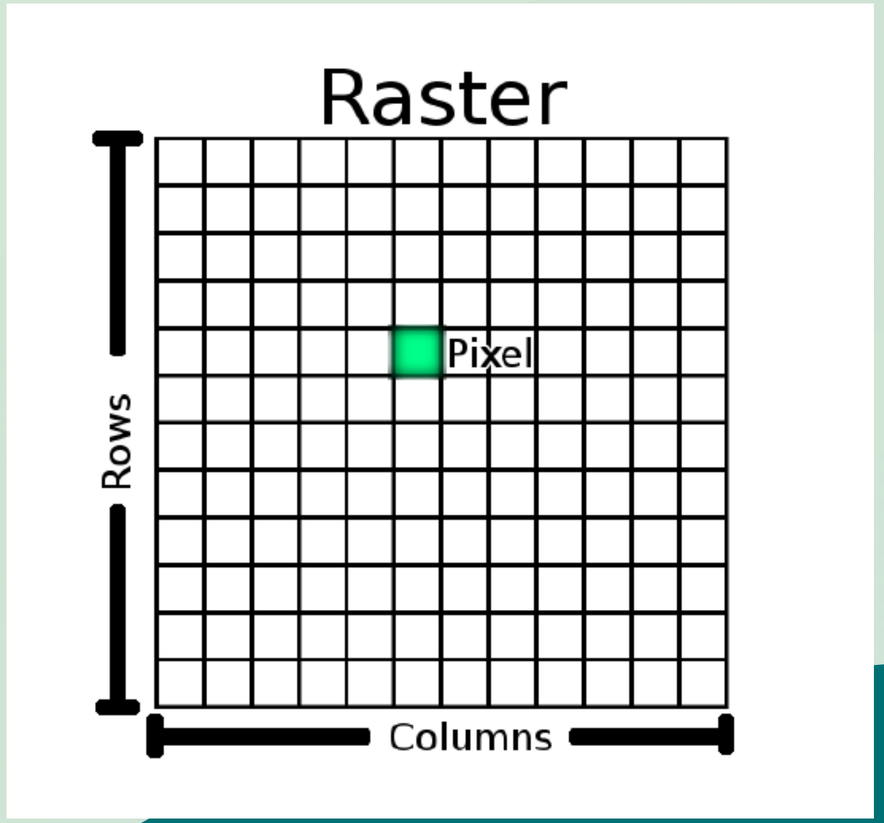
How images work

Digital images consist of a rectangular grid of pixels where each pixel is represented by a number or a set of numbers, describing its color in a given color space. These pixels are displayed horizontally, **row by row**.

These are called **raster or bitmap graphics**. Each pixel in the image is assigned a specific value that represents its color. In an **8-bit color** depth system, the pixel value ranges from **0 to 255**, where each value corresponds to a specific color in the color palette.

So there are totally **3 bytes** of color information per pixel.
This format copies holds for all pixels in the resolution of the image.

Each image format uses one of the available color depths combined with some level of compression (as in **JPEG**). PNG images have an extra channel called **alpha** to indicate transparency



Different steganographic techniques

- **LSB (Least Significant Bit) Steganography: It involves replacing the least significant bits of the pixel values in an image with the bits of the hidden message.**

Figure 2.7: The blue color on the left has the RGB values of 0, 162 and 232 respectively. The blue on the right has had the blue value slightly altered to 233.

Steps:

- 1. Convert the message into bits.
- 2. Select a color byte from a cover image pixel.
- 3. Substitute the least significant bit of that byte with a bit from the
- message.
- 4. Repeat 2. and 3. step until the whole message is hidden.

Pixel Value Differencing

Pixel Value Differencing (PVD) is a steganographic technique used to embed secret information within digital images. The goal of PVD is to hide data in such a way that it is imperceptible to the human eye while minimizing the impact on the image quality.

Here's an explanation of how Pixel Value Differencing works:

1. **Image selection:** First, an original cover image is selected, which serves as the carrier for the hidden information.
2. **Pixel value calculation:** Each pixel in the cover image is represented by a numerical value based on its color intensity. In grayscale images, the pixel value typically ranges from 0 (black) to 255 (white). In color images, pixel values are often represented using multiple color channels, such as red, green, and blue (RGB).
3. **Embedding process:** The secret data to be hidden (referred to as the payload) is typically a binary sequence of 0s and 1s. PVD works by manipulating the pixel values to embed the payload.
    a. Pixel pair selection
    b. Pixel value difference calculation
    c. Payload bit embedding
    d. Pixel value difference calculation

Different steganographic techniques

- **Spatial Domain Steganography: This technique focuses on altering the spatial domain properties of an image t information. It can involve techniques such as modifying the color values, changing the pixel positions, or app visual transformations to embed the hidden message.**

- **Transform Domain Steganography: Transform domain steganography utilizes mathematical transforms like the Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT) to embed information. The hidden mes is encoded within the transform coefficients, taking advantage of their frequency representations.**

Code Snippets

```python
def mask_n_bit_of_image(img_array, mask):
    """
    Applies a mask bitwise on an image to make the n lowest bit zero
    :param img: input image
    :param mask: mask to make the n lowest significant bits zero. Maske sample: int('11111110', 2)
    :return: masked image
    """
    for i in range(img_array.shape[0]):
        for j in range(img_array.shape[1]):
            new_value = img_array[i, j] & mask
            img_array[i, j] = new_value

    return img_array
```

```python
def image_binary_content(input_array):
    """
   Calculates the binary content of an input numpy array of type int.
   :param input_array: input numpy array which is a gray_scale image
   :return: binary content of the image in str format
   """


    img_cp = []
    for x in range(0, input_array.shape[0]):
        for y in range(0, input_array.shape[1]):
            img_cp.append(bin(int(input_array[x, y]))[2:])

    # reshaping the list to match the image size and order
    new_img_arr = np.reshape(img_cp, (input_array.shape[0], input_array.shape[1]))
    return new_img_arr
```

```python
def padding_zeros_to_make_8bits_images(input_image):
    """
    Checks the output of image_binary_content(img) to add zeros to the left hand side of every byte.
    It makes sure every pixel is represented by 8 bytes
    :param input_image: input image or numpy 2D array
    :return: numpy 2D array of 8-bits pixels in binary format
    """
    for i in range(input_image.shape[0]):
        for j in range(input_image.shape[1]):
            if len(input_image[i, j]) < 8:
                # print(input_image[i, j])
                zeros_to_pad = 8 - len(input_image[i, j])
                # print('Zeros to pad is {}'.format(zeros_to_pad))
                elm = input_image[i, j]
                for b in range(zeros_to_pad):
                    elm = '0' + elm
                # print('New value is {} '.format(elm))
                input_image[i, j] = elm
                # print('double check {} '.format(input_image[i, j]))

    return input_image
```

Applications

- Confidential Communication: Image steganography can be used to hide confidential data within images, enabling secure transmission over insecure channels such as the internet.

- Digital Watermarking: Image steganography can be employed for embedding watermarks in digital images to protect intellectual property rights.

- Information Concealment: Image steganography can be used to hide sensitive or confidential data within images, such as text documents, audio files, or other images.

Covert Surveillance: Steganography can be utilized to embed video or audio recordings within seemingly innocuous images.

Social Media Privacy:Private Messaging: Image steganography can be used for private messaging on social media platforms.

Conclusion

In conclusion, this project has explored the applications of image steganography in OpenCV, highlighting its significance in various domains such as data security, privacy, covert communication, digital forensics, and authentication. OpenCV, with its extensive range of image processing functions, provides a robust platform for implementing steganography techniques.

Future Work

Regarding the standard usage of steganography, one class of image steganography, called frequency domain steganography, uses changes in the frequency domain of the image to encode data. A variation of some such algorithm could be used to successfully to encode a texture and decode the message from a screenshot, though a hurdle still remains in the variable size of the screenshot relative to the original texture.

Thank you!