Bangalore Institute of Technology

Department of Computer Science and Engineering
K R Road, V V Pura, Bengaluru-560004

STEGANOGRAPHY

Submitted as the **Content Beyond the Syllabus** for the subject
Computer Graphics and Visualization (18CS62)

Submitted by

Shivani	**1BI20CS160**
Ullas B R	**1BI20CS182**
Vaitheeswaran J	**1BI20CS183**
Yatish Gowda	**1BI20CS194**

For academic year 2022-23

Under the guidance of
Dr. Gunavathi H S
Assistant Professor

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnanasangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V.V.Pura, Bangalore-560 004

Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of **CGV CONTENT BEYOND SYLLABUS** entitled "**Steganography**" has been successfully completed by

Shivani	**1BI20CS160**
Ullas B R	**1BI20CS182**
Vaitheeswaran J	**1BI20CS183**
Yatish Gowda	**1BI20CS194**

of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in Computer Science & Engineering of the Visvesvaraya Technological University during the academic year 2022-2023.

In-charge Faculty :

Dr. Gunavathi H S

Assistant Professor

Dept. of CS&E

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Project Overview

Our project focuses on implementing steganography techniques using the OpenCV Python library. Steganography is the practice of concealing information within an image or a video without raising suspicion. By leveraging the powerful capabilities of OpenCV, we aim to develop an efficient and secure method for embedding secret messages within digital media. Through this project, we plan to explore various steganographic algorithms, such as LSB (Least Significant Bit) and DCT (Discrete Cosine Transform), and their integration with OpenCV.

OpenCV: OpenCV, a widely used computer vision library in Python, provides a powerful set of tools for steganography. Its image processing capabilities enable seamless embedding and extraction of hidden messages within digital media. Combined with the flexibility and simplicity of Python, it offers an ideal environment for developing steganographic solutions with ease.

Steganography: Steganography is the practice of concealing secret information within seemingly innocuous digital media, such as images or videos. It involves embedding the hidden data in a way that is imperceptible to the human eye. Steganography plays a crucial role in secure communication, covert messaging, and protecting sensitive information from detection or interception.

1.2 Problem Statement

The aim of this project is to address the need for a reliable and user-friendly steganography solution using the OpenCV Python library. Existing steganographic techniques often lack efficiency, security, and ease of use, limiting their practical application. Our project seeks to overcome these limitations by developing a robust and efficient method for embedding secret messages within digital media. We aim to provide a solution that is accessible to users without extensive technical knowledge, while maintaining the integrity and visual quality of the cover media. By addressing these challenges, we strive to enable individuals to securely communicate and share information using steganography techniques implemented with the OpenCV Python library.

2. CONCEPT/ALGORITHM

2.1 Concept Of Steganography

Steganography is the practice of concealing secret information within a carrier medium in such a way that it appears innocuous and attracts minimal attention. Unlike cryptography, which focuses on making the content of a message unintelligible to unauthorized parties, steganography aims to hide the existence of the message itself.

The concept of steganography dates back centuries and has been used in various forms throughout history. The underlying principle is to embed the secret information within a seemingly ordinary object or communication medium, making it difficult for anyone to detect the presence of hidden data.

Here are some key aspects of steganography:

Carrier Medium: The carrier medium is the object or communication channel used to hide the secret message. It can be anything that can store or transmit data, such as images, audio files, videos, text documents, network packets, or even physical objects like paintings or printed documents.

Hidden Information: The hidden information, also known as the secret message or payload, is the content that needs to be concealed. It can be text, images, audio, video, or any other form of digital data. The size and nature of the hidden information can vary based on the capacity and characteristics of the carrier medium.

Embedding Technique: Steganography techniques involve embedding the secret information into the carrier medium in a way that it remains hidden to casual observers. Various embedding techniques can be employed, such as modifying the least significant bits of data elements, using unused areas of a file, or applying transformations to hide information in frequency or spatial domains.

Extraction and Decoding: To retrieve the hidden information, the recipient or authorized party must know the steganographic technique used and apply the corresponding extraction or decoding process. This process reverses the embedding technique to extract the concealed data from the carrier medium.

2.2 Least Significant Bit Algorithm (Lsb)

The LSB (Least Significant Bit) algorithm is a steganographic technique that involves hiding information within the least significant bits of digital data. It is commonly used to embed secret messages within digital images, audio files, or other digital media.

The secret message is divided into bits, and each bit is sequentially embedded into the LSB of the carrier file. Since the LSB has the least impact on the overall value of a data element, modifying it usually results in minimal noticeable changes to the carrier file.

Figure 2.1: 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.

To illustrate the LSB algorithm, let's consider an example using an 8-bit grayscale image. Each pixel in the image is represented by a single byte, which has 8 bits. The LSB of each byte is the rightmost bit, and it carries the least weight in determining the pixel's value.

10101011 ← LSB

Embedding Phase:

1. Convert the secret message into binary representation.
2. Traverse through the pixels of the carrier image.
3. For each pixel, replace the LSB with a bit from the secret message, moving from the most significant bit to the least significant bit.Repeat this process until all the bits of the secret message have been embedded or until the desired capacity is reached.

Extraction Phase:

1. Traverse through the pixels of the modified carrier image.
2. Extract the LSB of each pixel.

3. Combine the extracted bits to reconstruct the hidden secret message.

It's important to note that while the LSB algorithm is relatively simple, it has limitations. The embedding capacity is directly related to the size of the carrier file and the number of LSBs used. Higher embedding capacities may lead to more noticeable changes in the carrier file, potentially alerting someone to the presence of hidden information. Additionally, LSB-based steganography can be vulnerable to statistical analysis attacks, where patterns or anomalies in the LSBs are detected.

2.3 Code

```
import numpy as np

import cv2 as cv

import os

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

def draw_img_side_by_side(img1, img2, caption):
    h_im = cv.hconcat([img_cp, img])
    cv.imshow(caption, h_im)
```

```
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
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
def write_img(path, name, img):
    """
    :param path:
    :param name:
    :param img:
    :return:
    """
    name = os.path.join(path, name)
    cv.imwrite(name, img)
img_path = 'nier5.bmp'
img = cv.imread(img_path, 0)
cv.imshow('original image', img)
img_cp = img.copy()
path_dest = r'color'
```

```
print('Original image shape {}'.format(img.shape))

mask = int('11111100', 2)

print('mask = {}'.format(mask))

img_n2 = mask_n_bit_of_image(img, mask)

# draw_img_side_by_side(img_cp, img_n2, 'Modified image n=2')

img_to_hide_path = 'vinyl.jpg'

img_to_hide = cv.imread(img_to_hide_path, 0)

img_to_hide = cv.resize(img_to_hide, (220, 220), interpolation=cv.INTER_NEAREST)


cv.imshow('hidden image', img_to_hide)


h_flat = img_to_hide.flatten()

print('LENGTH OF FLAT HIDDEN IMAGE IS {}'.format(len(h_flat)))

# for i in range(len(h_flat)):

#     print(bin(h_flat[i]))


img_hidden_bin = image_binary_content(img_to_hide)

print('binary of hidden image type: {}'.format(type(img_hidden_bin)))

# reformat evey byte of the hidden image to have 8 bits pixels

img_hidden_bin = padding_zeros_to_make_8bits_images(img_hidden_bin)

print(img_hidden_bin.shape)


all_pixels_hidden_img = img_hidden_bin.flatten()
```

```
print('Length of flattened hidden image to embed is {}'.format(len(all_pixels_hidden_img)))
# for i in range(0, 48400):
#     print(all_pixels_hidden_img[i])


num_pixels_to_modify = len(all_pixels_hidden_img) * 4
print('Number of pixels to modify in base image is {}'.format(num_pixels_to_modify))
# parts = [your_string[i:i+n] for i in range(0, len(your_string), n)]
two_bit_message_list = []
for row in all_pixels_hidden_img:
    for i in range(0, 8, 2):
        two_bit_message_list.append(row[i: i+2])
print('TWO BITS MESSAGE LIST LENGTH {}'.format(len(two_bit_message_list)))
# insert 6 zeros to left hand side of every entry to two_bit_message_list
new_hidden_image = []
for row in two_bit_message_list:
    row = '000000' + row
    new_hidden_image.append(row)


base_img_flat = img_cp.flatten()
num_bytes_to_fetch = len(two_bit_message_list)
img_base_flat = img_n2.flatten()
print('LENGTH OF TWO BIT MSG LIST {}'.format(num_bytes_to_fetch))


print('Bit length of the bytes to fetch is {} '.format(bin(num_bytes_to_fetch)))
```

```
# scanned from new constructed image
print(bin(num_bytes_to_fetch)[2:])
print(len( bin(num_bytes_to_fetch)[2:] ))
print('Start of loop to embed the hidden image in base image')
for i in range(num_bytes_to_fetch):
    # First 12 bytes are reserved for the hidden image size to be embedded
    new_value = img_base_flat[i] | int( new_hidden_image[i], 2)
    img_base_flat[i] = new_value
image_with_hidden_img = img_base_flat.reshape(img_n2.shape)
cv.imshow('Image with hidden image embedded', image_with_hidden_img)


# Reading embedded image from constructed image
constructed_image_with_message_embedded =
image_binary_content(image_with_hidden_img)
constructed_image_with_message_embedded_zero_padded =
padding_zeros_to_make_8bits_images(constructed_image_with_message_embedded)
flat_constructed_image_with_message_embedded =
constructed_image_with_message_embedded_zero_padded.flatten()
embedded_img_list = []
for i in range(num_bytes_to_fetch):
    embedded_img_list.append(flat_constructed_image_with_message_embedded[i][-2:])


# [print(rec) for rec in embedded_img_list]
print('EMBEDDED IMAGE LIST LENGTH {}'.format(len(embedded_img_list)))
```

```
const_byte_list = []

for i in range(0, len(embedded_img_list), 4):

    const_byte = embedded_img_list[i] + embedded_img_list[i+1] + embedded_img_list[i+2]
+ embedded_img_list[i+3]

    const_byte_list.append(const_byte)


# [print(rec) for rec in const_byte_list]

print('LENGTH OF CONSTRUCT BYTES IS {}'.format(len(const_byte_list)))


const_byte_list_tmp = np.array(const_byte_list, np.float64)

const_byte_2D_array = const_byte_list_tmp.reshape(img_to_hide.shape)  #((220,220))

const_byte_2D_array = const_byte_2D_array.astype('uint16')

cv.imshow('Constructed image from base', const_byte_2D_array)

cv.imwrite('reconstructed_image.jpeg', const_byte_2D_array)


cv.waitKey(0)

cv.destroyAllWindows()
```

3. RESULT

3.1 Result

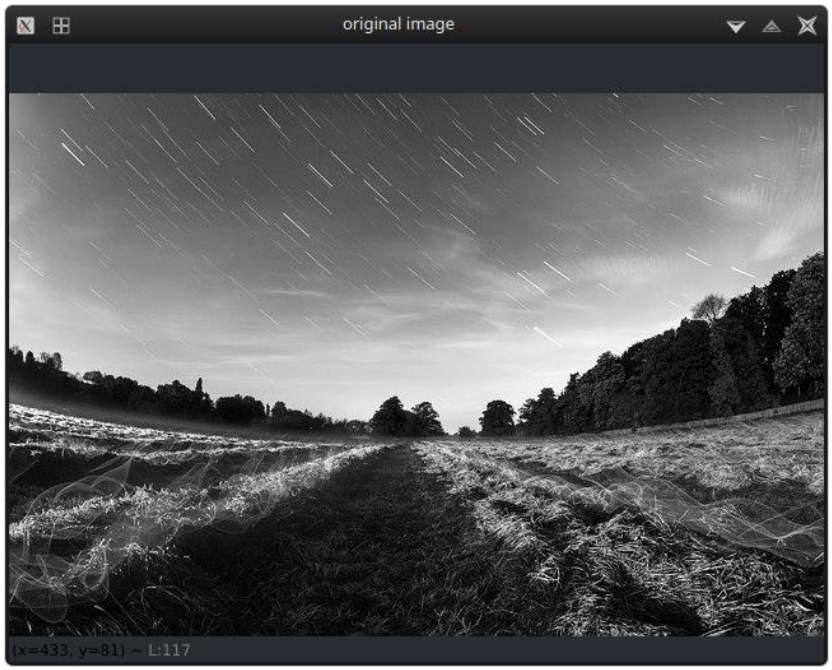

Figure 3.1: Cover Image

Figure 3.2: Hidden Image

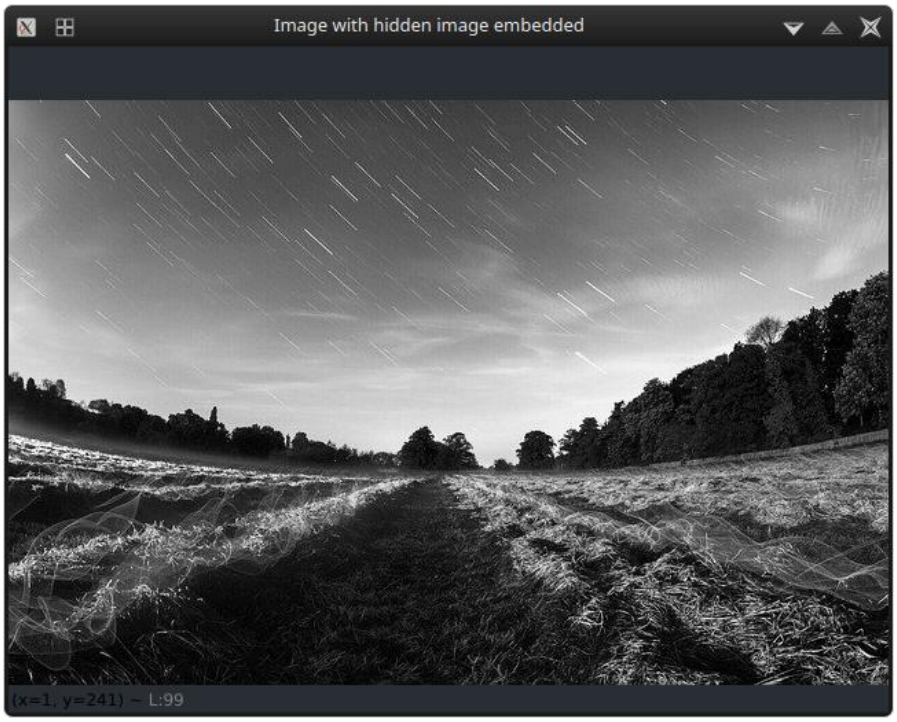

Figure 3.3: Cover Image Embedded with hidden image

Figure 3.4: Constructed Image

4. APPLICATIONS

4.1 Applications

Confidential Communication:

Secure Data Transmission: Image steganography can be used to hide confidential data within images, enabling secure transmission over insecure channels such as the internet. The hidden information can only be extracted by the intended recipient.

Covert Messaging: Steganography allows for secret communication by embedding hidden messages within images. This can be useful in scenarios where overt communication is monitored or restricted.

Digital Watermarking:

Copyright Protection: Image steganography can be employed for embedding watermarks in digital images to protect intellectual property rights. Watermarks can be hidden within the image, making it difficult to remove or alter without detection.

Data Hiding:

Information Concealment: Image steganography can be used to hide sensitive or confidential data within images, such as text documents, audio files, or other images. This technique provides an additional layer of security by making the hidden data inconspicuous.

Covert Surveillance:

Secret Recording: Steganography can be utilized to embed video or audio recordings within seemingly innocuous images. This enables covert surveillance and discreet capture of information without arousing suspicion.

Digital Forensics:

Evidence Protection: Steganography can aid in preserving digital evidence by concealing sensitive information within images. This ensures the integrity and confidentiality of evidence during investigations

Steganalysis: Steganography techniques can be employed to detect and extract hidden data from images, facilitating the analysis of potentially malicious activities or unauthorized data exchanges.

Social Media Privacy:

Private Messaging: Image steganography can be used for private messaging on social media platforms. By embedding hidden messages within shared images, users can communicate securely without drawing attention.

Authentication and Anti-counterfeiting:

Image Integrity Verification: Steganography can help verify the authenticity and integrity of images by embedding digital signatures or cryptographic hashes within them. Any tampering or unauthorized modifications can be detected through the extracted hidden information.

Anti-counterfeiting Measures: Steganography techniques can be employed to embed hidden security features within product images or packaging, aiding in counterfeit detection and prevention.

Research and Education:

Steganography Analysis: Image steganography provides an interesting field of research for studying various embedding techniques, detection methods, and countermeasures.

Educational Purposes: Studying image steganography in practical applications can enhance understanding of data security, image processing, and information hiding techniques.

5. CONCLUSION

5.1 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.

Through the utilization of image steganography, sensitive information can be concealed within digital images, ensuring secure data transmission and confidential communication. Steganography also plays a crucial role in digital watermarking, protecting intellectual property rights by embedding hidden watermarks within images.

The applications of image steganography in OpenCV extend to covert surveillance, where video or audio recordings can be hidden within images for discreet capture of information. In the realm of digital forensics, steganography aids in evidence protection and analysis, enabling the detection and extraction of hidden data from images.

Social media privacy can be enhanced using image steganography, allowing for private messaging and secure communication through hidden messages embedded within shared images. Additionally, steganography techniques contribute to image integrity verification and anti-counterfeiting measures, facilitating image authentication and protecting against counterfeit products.

This project has shed light on the potential use cases of image steganography in OpenCV, showcasing its versatility and significance in ensuring data security, privacy, and authentication in various real-world scenarios. However, it is essential to acknowledge the limitations and potential vulnerabilities associated with steganography techniques, necessitating ongoing research and the implementation of appropriate security measure.

REFERENCES

References

[1] Fridrich, J.(2009) Steganography in Digital Media: Principles, Algorithms, and Applications. Cambridge University Press.

[2] Khan, M., & Huda, S. (2015). A Computer Study of Image Steganography Techniques. International Journal of Computer Applications, 119(4), 1-5.

[3] Cheddad, A., Condell, J., Curran, K. Digital Image Steganography: Survey and Analysis of Current Methods. Signal Processing.

[4] OpenCV Documentation Available at: https://docs.opencv.org/