**CODE IMPLEMENTATION**

6.1 CODE SAMPLE:

""" Steganography is the process of hiding the secret data in the multimedia carrier,

Eg: image, video and audio files.

In this program we are going to Encode and Decode the secret data in an Image.... """

# Importing the packages which we are going to use.

import pyautogui

import cv2

import numpy as np

import types

from tkinter import messagebox as mb

from tkinter import \*

# The steganography functions starts from here.

# Creating the encoding function.

def encode\_text(image\_name , data , filename):

image = cv2.imread(image\_name) # Read the input image using OpenCV-Python.

# OpenCV is a package to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

if (len(data) == 0):

raise ValueError('Data is empty')

encoded\_image = Encode(image, encMessage(data)) # Call the Encode function to hide the secret message into the selected image.

cv2.imwrite(filename, encoded\_image) # The final image will be saved at the destination path.

def encMessage(data1):

alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

special\_char = " ‘!@$#%^&\*(),./;:"

numbers = "1234567890"

add = 3

new\_word = ""

for i in data1:

position = alphabet.find(i)

if i in special\_char or i in numbers:

new\_word += i

elif (position + add) > 51:

m = position + add - 51

new\_word += alphabet[m - 1]

else:

new\_word += alphabet[position + add]

return new\_word

# This function will convert the message to binary format.

def messageToBinary(message):

if type(message) == str:

return ''.join([format(ord(i), "08b") for i in message])

elif type(message) == bytes or type(message) == np.ndarray:

return [format(i, "08b") for i in message]

elif type(message) == int or type(message) == np.uint8:

return format(message, "08b")

else:

raise TypeError("Input type not supported")

# Function to hide the Secret data in the Image pixels.

def Encode(image, secret\_message):

# Calculate the maximum bytes to encode.

n\_bytes = image.shape[0] \* image.shape[1] \* 3 // 8

# Check if the number of bytes to encode is less than the maximum bytes in the image.

if len(secret\_message) > n\_bytes:

raise ValueError("Error encountered insufficient bytes, need bigger image or less data !!")

secret\_message += "//#//" # You can use any string as the delimeter.

data\_index = 0

# Convert input data to binary format using messageToBinary() fucntion.

binary\_secret\_msg = messageToBinary(secret\_message)

data\_len = len(binary\_secret\_msg) # Find the length of data that needs to be hidden.

for values in image:

for pixel in values:

# Convert RGB values to binary format.

r, g, b = messageToBinary(pixel)

# Modify the least significant bit only if there is still data to store.

if data\_index < data\_len:

# Hide the data into least significant bit of red pixel.

pixel[0] = int(r[:-1] + binary\_secret\_msg[data\_index],2)

data\_index += 1

if data\_index < data\_len:

# Hide the data into least significant bit of green pixel.

pixel[1] = int(g[:-1] + binary\_secret\_msg[data\_index],2)

data\_index += 1

if data\_index < data\_len:

# Hide the data into least significant bit of blue pixel.

pixel[2] = int(b[:-1] + binary\_secret\_msg[data\_index],2)

data\_index += 1

# If data is encoded, just break out of the loop.

if data\_index >= data\_len:

break

return image

def Decode(image):

binary\_data = ""

for values in image:

for pixel in values:

r, g, b = messageToBinary(pixel) # Convert the red,green and blue values into binary format.

binary\_data += r[-1] # Extracting data from the least significant bit of red pixel.

binary\_data += g[-1] # Extracting data from the least significant bit of green pixel.

binary\_data += b[-1] # Extracting data from the least significant bit of blue pixel.

# Split by 8-bits.

all\_bytes = [binary\_data[i: i + 8] for i in range(0, len(binary\_data), 8)]

# Convert from bits to characters.

decoded\_data = ""

for byte in all\_bytes:

decoded\_data += chr(int(byte, 2))

if decoded\_data[-5:] == "//#//": # Check if we have reached the delimeter which is "#####".

break

return decoded\_data[:-5] # Remove the delimeter to show the original hidden message.

def decMessage(data1):

alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

special\_char = " ‘!@$#%^&\*(),./;:"

numbers = "1234567890"

add = -3

new\_word = ""

for i in data1:

position = alphabet.find(i)

if i in special\_char or i in numbers:

new\_word += i

elif (position + add) > 51:

m = position + add - 51

new\_word += alphabet[m - 1]

else:

new\_word += alphabet[position + add]

return new\_word

# Creating a decoding function.

def decode\_text(image\_name , varchar):

# Read the image that contains the hidden image

image = cv2.imread(image\_name) # Read the image using cv2.imread()

text1 = Decode(image)

text2 = decMessage(text1)

#Set the value for the varchar variable that is used to show the data in the GUI

varchar.set(text2)

#Creating the GUI for the encoding function

def encode\_image():

root.destroy()

encode\_wn = Tk()

encode\_wn.title("Encoding........")

encode\_wn.geometry('600x220')

encode\_wn.resizable(0, 0)

encode\_wn.config(bg='#FFE4B5')

Label(encode\_wn, text='Encode an Image', font=("Georgia", 15), bg='#FFE4B5').place(x=220,y=10, rely=0)

Label(encode\_wn, text='Enter the path to the image(with extension):', font=("Times New Roman", 13),

bg='#FFE4B5').place(x=10, y=50)

Label(encode\_wn, text='Enter the data to be encoded:', font=("Times New Roman", 13), bg='#FFE4B5').place(

x=10, y=90)

Label(encode\_wn, text='Enter the output path and file name (with extension):', font=("Times New Roman", 12),

bg='#FFE4B5').place(x=10, y=130)

img\_path = Entry(encode\_wn, width=35)

img\_path.place(x=350, y=50)

text\_to\_be\_encoded = Entry(encode\_wn, width=35)

text\_to\_be\_encoded.place(x=350, y=90)

after\_save\_path = Entry(encode\_wn, width=35)

after\_save\_path.place(x=350, y=130)

def Encoder():

Response= mb.askyesno("PopUp","Do you want to encode the image?")

if Response == 1:

encode\_text(img\_path.get(), text\_to\_be\_encoded.get(), after\_save\_path.get())

mb.showinfo("Pop Up","Successfully Encoded\nPlease close the encoding tab to avoid Duplicates of the Encoded image.")

else:

mb.showwarning("Error...","Unsuccessful, please try again")

Button(encode\_wn, text='Encode the Image', font=('Times New Roman', 12), bg='#BBFFFF', command=lambda:

Encoder()).place(x=220, y=175)

#Creating the GUI for Decoding function

def decode\_image():

root.destroy()

decode\_wn = Tk()

decode\_wn.title("Decoding.......")

decode\_wn.geometry('900x300')

decode\_wn.resizable(0, 0)

decode\_wn.config(bg='#FFE4B5')

Label(decode\_wn, text='Decode an Image', font=("Geogia", 15), bg='#FFE4B5').place(x=350,y=10, rely=0)

Label(decode\_wn, text='Enter the path to the image (with extension):', font=("Times New Roman", 16),

bg='#FFE4B5').place(x=30, y=50)

img\_entry = Entry(decode\_wn, width=65)

img\_entry.place(x=450, y=55)

text\_varchar = StringVar()

Button(decode\_wn, text='Decode the Image', font=('Times New Roman', 12), bg='#BBFFFF', command=lambda:

decode\_text(img\_entry.get() , text\_varchar)).place(x=360, y=90)

Label(decode\_wn, text='Text that has been encoded in the image', font=("Times New Romam", 17), bg='#FFE4B5').place(

x=235, y=125)

text\_entry = Entry(decode\_wn, width=145, text=text\_varchar, state='readonly')

text\_entry.place(x=15, y=155, height=100)

# Initializing the window

#Main GUI for buttons Encode and Decode

root = Tk()

root.title('PROJECT WORK')

root.geometry('300x200')

root.resizable(0, 0)

root.config(bg='#FFE4B5')

Label(root, text='IMAGE \n STEGANOGRAPHY', font=('Georgia', 15),bg='#FFE4B5',

wraplength=300).place(x=50, y=10)

Button(root, text='Encode', width=25, font=('Times New Roman', 13), bg='#BBFFFF', command=encode\_image).place(

x=30, y=80)

Button(root, text='Decode', width=25, font=('Times New Roman', 13), bg='#BBFFFF', command=decode\_image).place(

x=30, y=130)

# Finalizing the window

root.update()

root.mainloop()

6.2 BIT PLANE SLICING:

import numpy as np

import cv2

img=cv2.imread('V:\Image Steganography\Examples\Encoding Image.jpg',0)

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing\Orginal Input Image.jpg',img)

cv2.waitKey(0)

lst = []

for i in range(img.shape[0]):

for j in range(img.shape[1]):

lst.append(np.binary\_repr(img[i][j], width=8))

eight\_bit\_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) \* 128).reshape(img.shape[0],img.shape[1])

seven\_bit\_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) \* 64).reshape(img.shape[0],img.shape[1])

six\_bit\_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) \* 32).reshape(img.shape[0],img.shape[1])

five\_bit\_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) \* 16).reshape(img.shape[0],img.shape[1])

four\_bit\_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) \* 8).reshape(img.shape[0],img.shape[1])

three\_bit\_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) \* 4).reshape(img.shape[0],img.shape[1])

two\_bit\_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) \* 2).reshape(img.shape[0],img.shape[1])

one\_bit\_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) \* 1).reshape(img.shape[0],img.shape[1])

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 7.jpg',cv2.normalize(eight\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 6.jpg',cv2.normalize(seven\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 5.jpg',cv2.normalize(six\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 4.jpg',cv2.normalize(five\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 3.jpg',cv2.normalize(four\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 2.jpg',cv2.normalize(three\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 1.jpg',cv2.normalize(two\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imwrite('V:\Image Steganography\Examples\Bit Plane Slicing/bit plane 0.jpg',cv2.normalize(one\_bit\_img , np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.waitKey(0)

#You can give any destination path for saving the images...

6.3 FULL CODE LINKS:

From these below links you can download full code and can try yourself implementing image steganography.

**MEGA:** https://mega.nz/folder/PeIE2LjA#ob4gWAI9qoUV9AY3EBNlNA

**GDRIVE:** https://drive.google.com/drive/folders/1HdcXqt-vNkQ3naQ0yTxYojsTp5AYVxXH?usp=sharing

**BLOGGER:** https://villainways.blogspot.com/2022/10/image-steganography.html