TABLE OF CONTENTS

INTRODUCTION

1.1 Problem Statement :

STEGANOGRAPHY FOR DIFFERENT PATTERN

1.2 Relevance of the project :

Steganography and Cryptography are excellent means to secure data in digital form. The purpose of steganography is covert communication to hide a message from a third party. This differs from cryptography, the art of secret writing, which is intended to make a message unreadable by a third party but does not hide the existence of the secret communication.

Although steganography is separate and distinct from cryptography, there are many analogies between the two, and some authors categorize steganography as a form of cryptography since hidden communication is a form of secret writing. While Cryptography has been implemented for years, Steganographic technologies are a very important part of the future of Internet security and privacy on open systems such as the Internet.

The importance of Steganography comes from the fact that there is no reliability over the medium through which the information is sent, in other words, the medium is not secured. Hence, this project is required to bring out the importance of steganography to prevent an unintended user from extracting any information.

1.3 Objective:

In the present day scenario of various internet connected devices, exchange of data that is secure and prevented from any unauthorised access is of very high importance. Steganography is a modern form of hiding secure messages or data within any digital media such as images and videos.

The objective of this project is to implement and show the working of Steganography for a digital image using python programming to encode, decode, and check hidden messages, in the form of

images, by minimum alteration of the RGB pixels using the PIL(Python Imaging Library), OpenCV API, and Tkinter GUI(Graphic User Interface).

Therefore, the main objective of this project is to hide and/or retrieve the required information behind digital images.

Few basic objectives:

- ➢ Encoding the desired message(data) into an image and saving an encoded image.
- ➢ Decoding hidden data from any given encoded image.
- ➢ Checking/Verifying whether two images are equal(To ensure encoded image is different from the original image)
- ➢ Provide a medium for steganography of a digital image through a desktop application.

SYSTEM REQUIREMENTS

2.1 Functional Requirements

Steganography of a digital image is done using python programming along with a graphic user interface to demonstrate and implement hiding of secret data in the form of digital images.

- ➔ User can have a choice from the menu : Encode, Decode, Check, and Exit
- ➔ Choosing Encode option from the menu will open a new window asking the user to enter the complete path of the file where the original image is saved, the message to be encoded and the full path of the file where the newly encoded image is to be saved.
- ➔ Leaving any of the entries in Encode window empty will prompt a pop-up window warning user to input entries.
- ➔ Clicking okay option on pop-up window closes it
- ➔ On successful encoding, a pop-up window with the respective message appears.
- ➔ Decode option from the menu will open a new window asking the user to enter the complete path of the image that is to be decoded to retrieve the hidden data.
- ➔ Empty field will open a pop-up window warning the user to input entries, clicking okay closes the pop-up window.
- ➔ On successful decoding, a pop-up window with the decoded data appears.
- ➔ Check option from the menu will open a new window asking the user to enter the complete paths of the images that are to be compared.
- ➔ Empty field will open a pop-up window warning the user to input entries, clicking okay closes the pop-up window.
- ➔ On successful checking, a pop-up window with the respective message(whether images have the same size, number of channels and if they are equal) appears.
- ➔ Exit option from the menu will stop execution.

2.2 Hardware and Software Requirements

2.2.1 Hardware Requirements

- **Processor: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz**
- **1800 Mhz**
- **4 Core(s)**
- **8 Logical Processor(s)**

2.2.2 Software Requirements

- **Operating system-Microsoft Windows 10**
- **Python 3.8**
- **Tkinter GUI(Graphic User Interface)**
- **OpenCV API**
- **PIL(Python Imaging Library)**

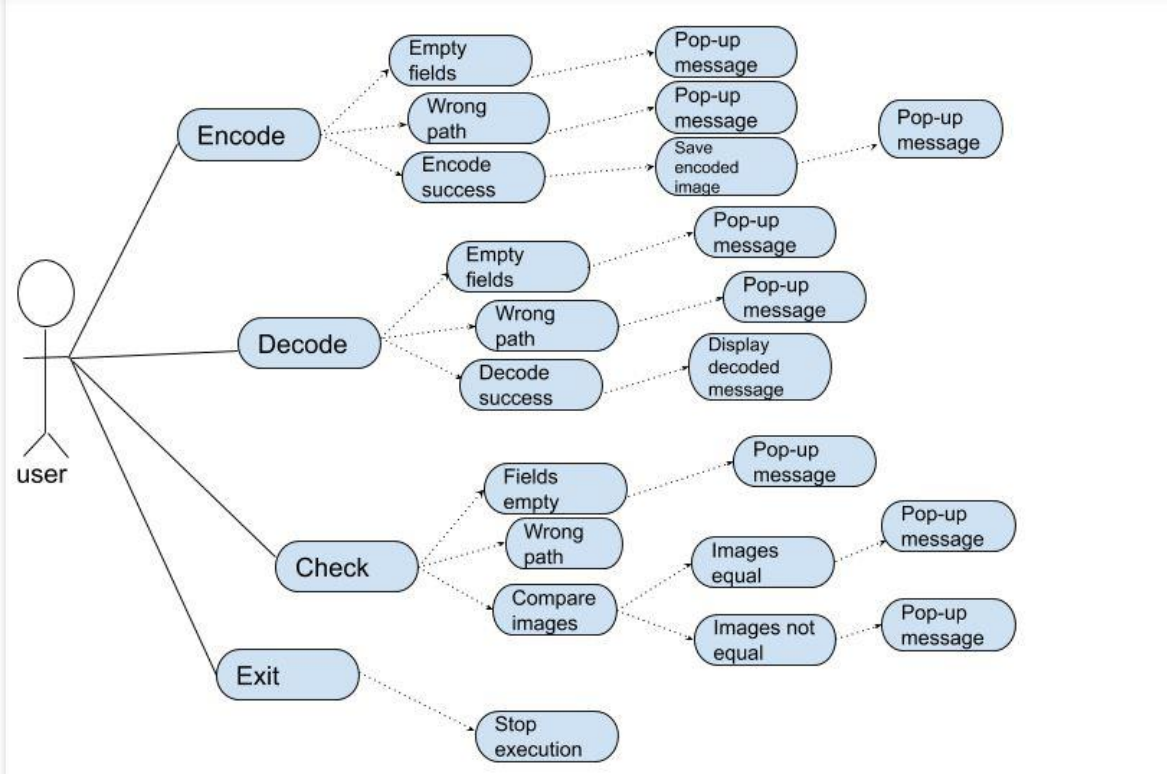
DESIGN AND IMPLEMENTATION

3.1 DESIGN :

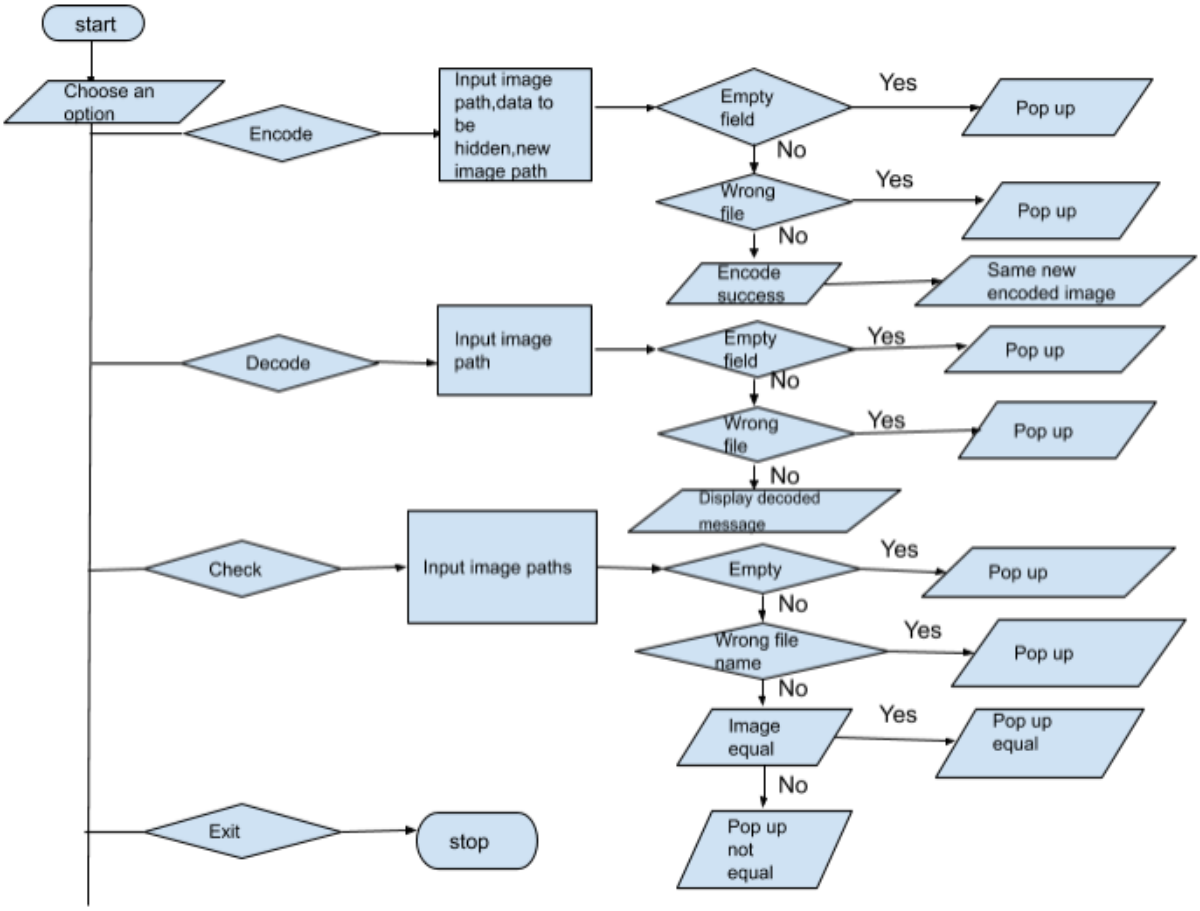
Steganography consists of several design components mentioned below:

1. MAIN WINDOW DESIGN
2. ENCODE WINDOW DESIGN
3. DECODE WINDOW DESIGN
4. CHECK WINDOW DESIGN

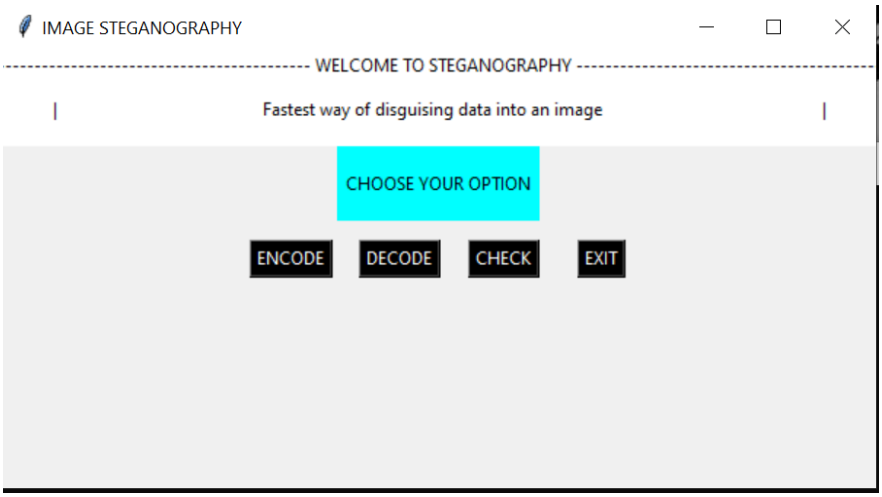
The use case diagram of the basic working is shown below :



The flowchart design of the basic working is shown below :

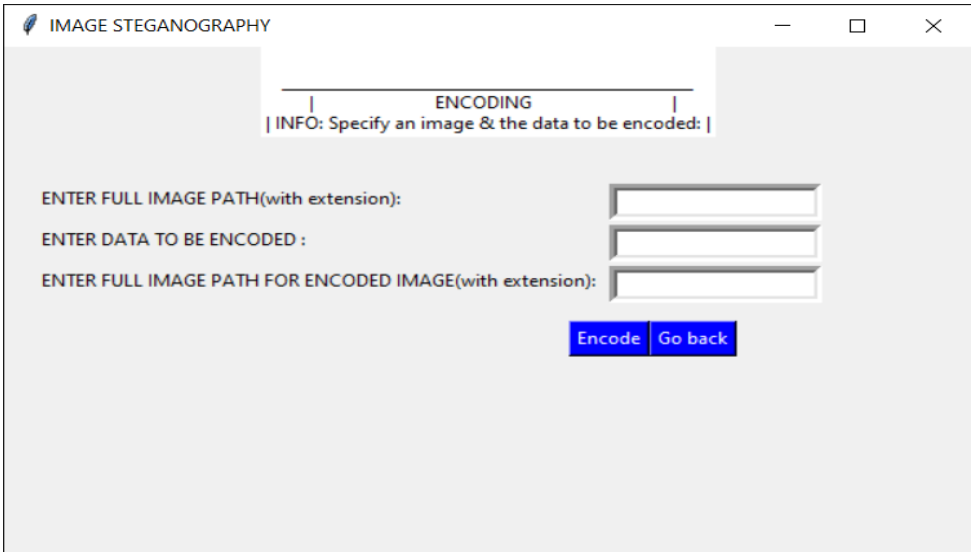


3.1.1 MAIN WINDOW DESIGN(without code):



The Main screen in the above figure was created as part of the first robust prototype. This is a windows form created with *Tkinter GUI(Graphic User Interface)*. As seen in the figure , the interface of the system, that the Main Screen is not constrained with heavy use of images or colours and it is concise and clear to understand as the buttons are labelled to make it easier to understand where the button navigates to.

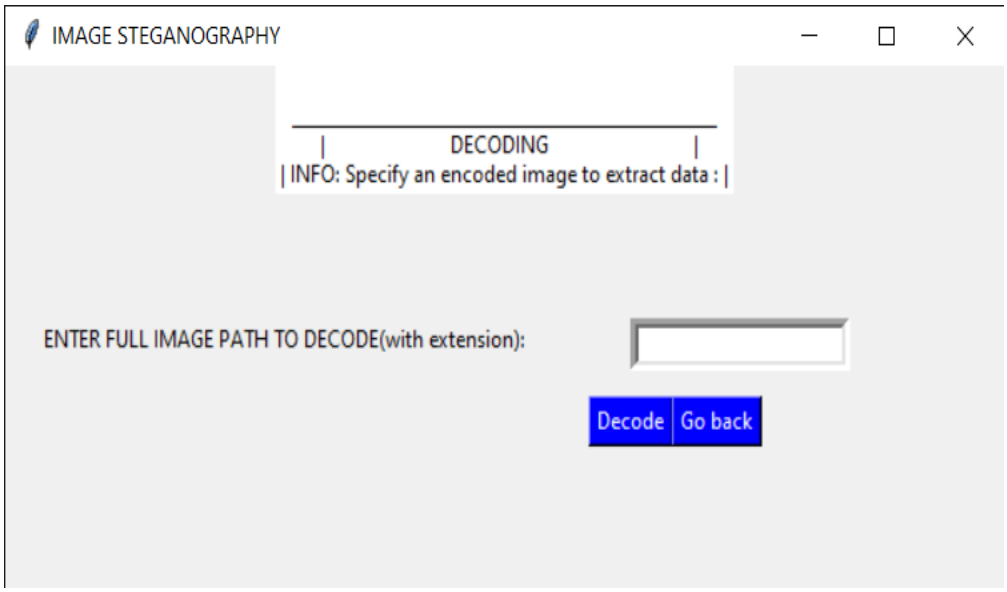
3.1.2 ENCODE WINDOW DESIGN(without code):



As seen in the above figure, The interface of the system is not constrained with heavy use of images or colours and it is clear to see and understand as the buttons are labelled to make it easier to understand where the button navigates to.

- There are altogether *three text fields* to be entered by the user in the Encode Screen:
  - Image path for the image to be encoded.
  - Data to be encoded in the image.
  - New image path for encoded image.
- In the above figure we can also see there are two buttons given:
  - *Encode button:* when pressed the given data would be encoded in the image provided by the user and the encoded image would be saved in the new path given by the user.
  - *GoBack button:* When pressed the user shall be taken back to the main screen.

3.1.3 DECODE WINDOW DESIGN(without code):

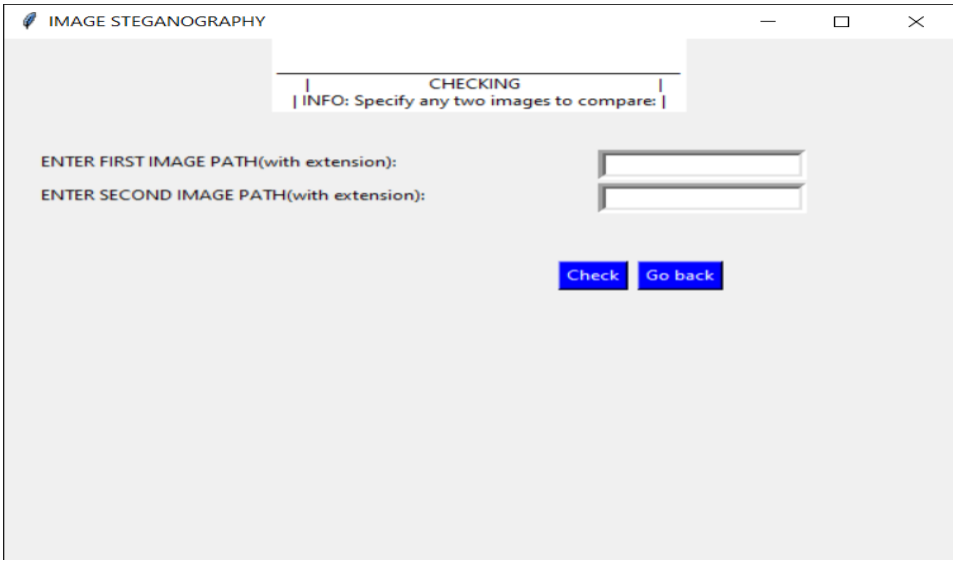


As seen in the figure, The interface is not constrained with heavy use of images or colours and it is clear to see and understand as the buttons are labelled to make it easier to understand where the button navigates to.

Decode Screen contains three components of one labeled text field along with two buttons:

- *Text field:* The user is provided with a text field for an encoded image path input that the user has to provide for decoding an image.
- *Buttons:*
    - *Decode button:* When pressed the given image from the image path is processed for decoding.
    - *GoBack button:* When pressed the user is taken back to the main screen.

3.1.4 CHECK WINDOW DESIGN(without code):



As seen in the figure, The interface is not constrained with heavy use of images or colours and it is clear to see and understand as the buttons are labelled to make it easier to understand where the button navigates to.

Check Screen mainly has four components of two labeled text fields and two buttons two successfully conduct a comparison of two images:

- *Text Fields:*
  - *First image path*: The user is asked to provide the first image file path to compare.
  - *Second image path*: The user is asked to provide the second image file path to compare.
- *Buttons:*
  - *Check button*: When pressed both images would be compared and a result is displayed.
  - *GoBack button:* When pressed the user would be taken back to the main window

3.2 IMPLEMENTATION :

The whole application has been implemented with three functionalities of encoding , decoding and checking for encoding data , decoding data & to find differences on the pixel level between two images.

3.2.1 ENCODING(with code):

The encoding functionality facilitates the methods for encrypting the user given data into the image by tapping into the image pixels triplets and changing the pixel value according to the data to be encoded.

★ *Encoding functionality(code):*

- The given data to be encoded is split into individual characters.
- The characters are converted into their binary equivalents and stored in a data list.
- The list of pixels of image that are stored in tuples of 3 values each(R,G,B channels) are accessed in groups of 3.
- Thus, 9 values in 3 groups of pixels are accessed for each character.
- The first 8 pixel values are altered based on the binary converted data where ‘0’ converts pixel value to even and ‘1’ converts pixel value to odd.
- The 9th bit is made even if data is to be read further and made an odd value if there is no more data to be encoded.
- The changed pixels are then updated onto the new image’s pixel data and the new encoded image is saved. This concludes the encoding.

```
def encode(img, data, new_img):
    try:
        image = PIL.Image.open(img , 'r')
        copy_img = image.copy() #creating copy to work on
        datalist = []
        for i in data:
        datalen = len(datalist)
        pixiter = iter(copy_img.getdata())  #list of pixel line by line

        pixellist=[] #storing new pixels here

            #encoding here
        for i in range(datalen):

            pixel = [value for value in next(pixiter)[:3]+ next(pixiter)[:3] + next(pixiter)[:3]]

                #changing pixel values
                #odd for 1 , even for 0
            for j in range(8):
                if (datalist[i][j] == '0' and pixel[j]%2 != 0):
                        #binary value of data is 0 and existing
                                #pixel is odd then make it even
                    pixel[j]-=1

                elif (datalist[i][j] == '1' and pixel[j]%2 == 0):
                        #binary is 1 and pixel is even then change to odd
                    pixel[j]+=1
          #end of j loop
                #whether to stop or read further at 9th bit
                #even to continue odd to stop
                #if even set to odd
            if(i == (datalen -1)):
                    #reached last bit
                if (pixel[-1]%2==0):  #even so set to odd
                    if(pixel[-1]!=0):
                        pixel[-1]-=1
                    else:
                        pixel[-1]+=1
            else:  #not reached end so set last bit to even
                if (pixel[-1]%2 != 0):
                    pixel[-1]-=1
                #replacing pixels
            new_pixel = tuple(pixel)
            (pix1,pix2,pix3) = (tuple(new_pixel[0:3]),tuple(new_pixel[3:6]),tuple(new_pixel[6:9]))
            pixellist.append(pix1)
            pixellist.append(pix2)
            pixellist.append(pix3)
    #end of i loop
        copy_img.putdata(pixellist)
            #saving encoded image
        popupmsg("ENCODING SUCCESSFUL...ENCODED IMAGE SAVED SUCCESSFULLY !!")
        copy_img.save(new_img, str(new_img.split(".")[1].upper()))
    except:
        popupmsg("ERROR!! FILE NOT FOUND! PLEASE TRY AGAIN")
```

3.2.2 DECODING(with code):

The Decoding functionality facilitates the method for decrypting the data from an encoded image. The method uses the same logic as the encoding algorithm, and taps into the changed pixels values to extract the encoded data from the given image.

- ★ *Decoding functionality(code):*
    - The given input image's pixel data is obtained and iterated in groups of 9 pixel values in groups of 3.

- The first 8 bits are accessed and the pixel values are converted an saved as binary ina separate data list.(if pixel is even, ‘0’ is appended to the data list, if pixel is odd, ‘1’ is appended to the data list and so on.)
- The 9th bit is checked so as to determine whether to continue reading pixels or not.(if last bit is even, continue reading pixels, if last bit is odd, stop reading.)
- The binary data is then converted to characters which retrieves the final decoded data. The decoded data is then displayed successfully. This concludes the decoding process.

```
def decode(img):
    try:
        image = PIL.Image.open(img, 'r')
        data = '' #entire data in list
        pixiter = iter(image.getdata())
        while True:
            #extracting 3 pixels at a time - 8 bit data 9th bit indicator
            pixel = [value for value in next(pixiter)[:3]+ next(pixiter)[:3] + next(pixiter)[:3]]
            binstr = '' #storing bin value
            for i in (pixel[:8]):
                if (i%2 == 0): #if even append 0
                    binstr+='0'
                else: #if odd append 1
                    binstr+='1'
            data+= chr(int(binstr,2)) #append converted char to data
            if (pixel[-1]%2 != 0): #if odd data is over
                return popupmsg("DECODING SUCCESSFUL...\nDECODED DATA =>    ",data)
    except:
        popupmsg("ERROR!! FILE NOT FOUND! PLEASE TRY AGAIN")
```

3.2.3 CHECKING(with code):

The Checking functionality facilitates the method for comparing two different images for similarities and differences on pixel level of the image. The Checking functionality is generally used to compare an image before and after encoding a set of data into the image.

The method uses the same logic of, tapping into the pixel triplets of both images and comparing them for any variations.

★ *Checking functionality(code):*

- The given input images are read and their shapes are compared.
- Reading the images gives the channel values, that is, the red, green and blue pixel values. The difference between the RGB values of the two images are calculated.
- The shape of the images tells us the size and the number of channels that the images have. If images have the same shape, they are said to be similar.
- If the shape of images are equal and the difference between the RGB channel values is zero, the images are said to be completely equal, else they are not equal. Thus the difference in pixel values determines if the images are exactly identical or not.

- If both criteria of similar shape, size and pixel values are met, then the images are completely equal, else they are not equal.

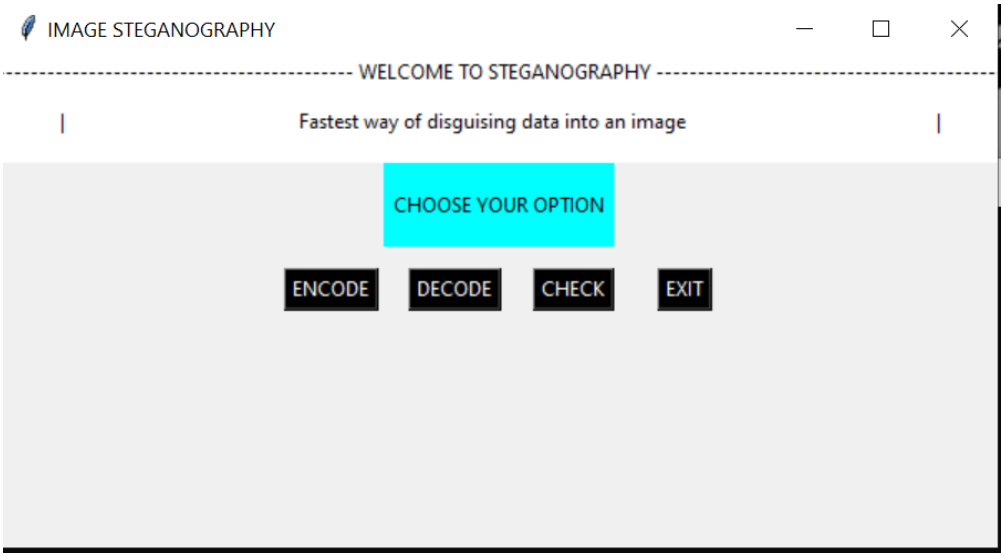
```
def check(img1, img2):
    #to check if images are equal
    try:
        original = cv2.imread(img1)
        duplicate = cv2.imread(img2)
        difference = cv2.subtract(original,duplicate) #difference in BGR pixels
        b, g, r = cv2.split(difference)
        if(original.shape == duplicate.shape):
            if(cv2.countNonZero(b) == 0 and cv2.countNonZero(g)== 0 and cv2.countNonZero(r) ==0): #gives height, width, channels(b,g,r)
                popupmsg("THE IMAGES HAVE SAME SIZE AND NUMBER OF CHANNELS AND THEY ARE COMPLETELY EQUAL")
            else:
                popupmsg("THE IMAGES HAVE SAME SIZE AND NUMBER OF CHANNELS BUT THEY ARE NOT COMPLETELY EQUAL")
        else:
            popupmsg("THE IMAGES DO NOT HAVE SAME SIZE AND NUMBER OF CHANNELS")
    except:
        popupmsg("ERROR !! FILE NOT FOUND ! PLEASE TRY AGAIN ")
```

TESTING AND DEPLOYMENT

With the completion of the system implementation stage, the system needs to be tested in-order to be reliable and stable. Various features in the system must integrate and work as planned so that correct results can be produced. To fulfil this aim, a testing plan has been planned and conducted strictly. The testing was done on the personal computer where the system was created and implemented.

- Testing has been divided into four categories:
  - MAIN WINDOW TESTING
  - ENCODING WINDOW TESTING
  - DECODING WINDOW TESTING
  - CHECKING WINDOW TESTING

4.1 MAIN WINDOW TESTING:

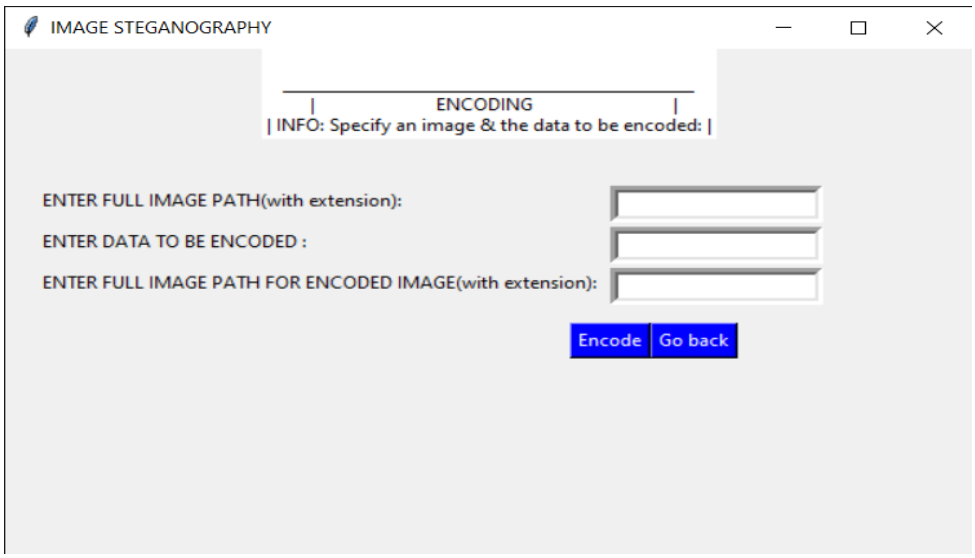


Test Results:

TEST ID	TEST DESCRIPTION	EXPECTED RESULT	OUTCOME
T1	ENCODE BUTTON	WHEN PRESSED THE USER SHOULD BE TAKEN TO A NEW WINDOW NAMELY: ENCODING WINDOW	PASS

T2	DECODE BUTTON	WHEN PRESSED THE USER SHOULD BE TAKEN TO A NEW WINDOW NAMELY: DECODING WINDOW	PASS
T3	CHECK BUTTON	WHEN PRESSED THE USER SHOULD BE TAKEN TO A NEW WINDOW NAMELY: CHECKING WINDOW	PASS
T4	EXIT BUTTON	WHEN PRESSED THE APPLICATION SHOULD CLOSE	PASS

4.2 ENCODING WINDOW TESTING:



Test Results:

TEST ID	TEST DESCRIPTION	EXPECTED RESULT	OUTCOME
T1	TEXTFIELD FOR IMAGE PATH THAT IS TO BE ENCODED	ACCEPT USER INPUT FOR IMAGE PATH.	PASS
T2	TEXTFIELD FOR DATA TO BE ENCODED	ACCEPT USER INPUT FOR DATA TO BE ENCODED.	PASS

T3	TEXTFIELD FOR NEW IMAGE PATH-ENCODED IMAGE.	ACCEPT USER INPUT FOR NEW IMAGE PATH WHERE ENCODED IMAGE IS TO BE SAVED.	PASS
T4	ENCODE BUTTON	WHEN PRESSED THE GIVEN DATA SHOULD BE ENCODED IN THE IMAGE AND NEW IMAGE MUST BE SAVED AT THE GIVEN IMAGE PATH	PASS
T5	GO BACK BUTTON	WHEN PRESSED USER SHOULD BE BROUGHT BACK TO THE MAIN WINDOW.	PASS
T6E1	ERROR POPUP-1	SHOULD POPUP WHEN THE USER LEAVES ANY OF THE ABOVE TEXT FIELDS EMPTY.	PASS
T7E2	ERROR POPUP-2	SHOULD POPUP ERROR MESSAGE WHEN THE GIVEN IMAGE PATH FOR THE IMAGE TO BE ENCODED IS NOT VALID.	PASS
T8S1	SUCCESSFUL POPUP	SHOULD POPUP A MESSAGE BOX WHEN ALL THE PROCESS HAS BEEN COMPLETED AND THE GIVEN DATA HAS BEEN SUCCESSFULLY ENCODED IN THE IMAGE AND SAVED AT THE NEW PATH GIVEN BY THE USER.	PASS

NOTE:

★ The snapshots to validate the *popup* test have been attached below.

- T6E1:

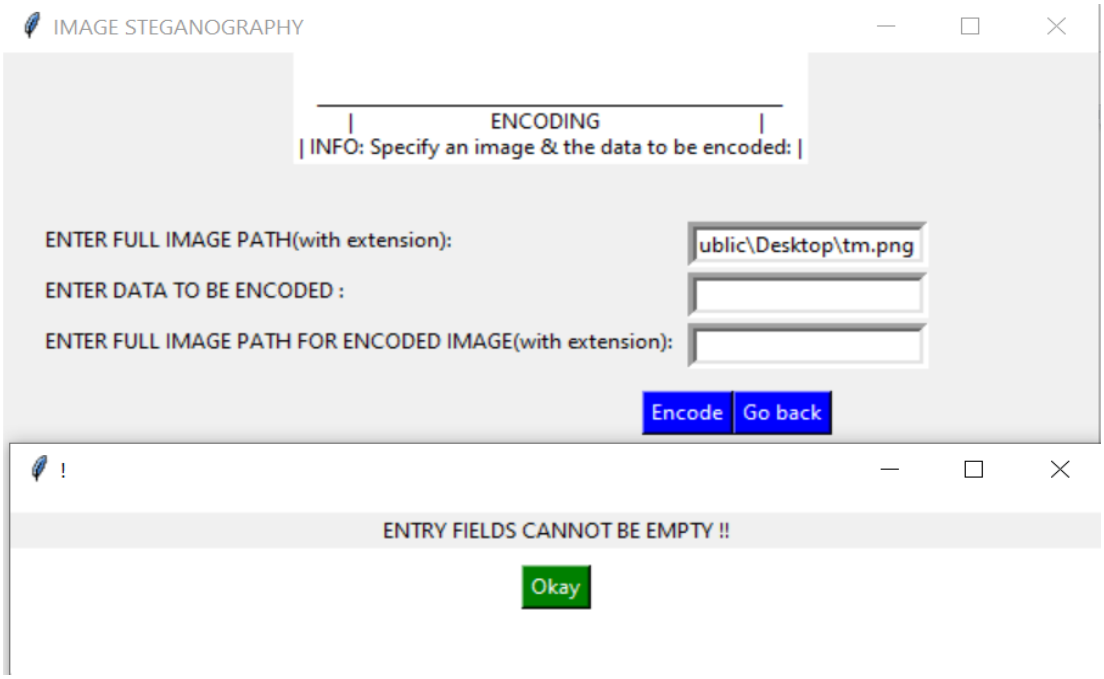


IMAGE STEGANOGRAPHY
ENCODING
| INFO: Specify an image & the data to be encoded: |
ENTER FULL IMAGE PATH(with extension):
ublic\Desktop\tm.png
ENTER DATA TO BE ENCODED :
ENTER FULL IMAGE PATH FOR ENCODED IMAGE(with extension):
Encode
Go back
!
ENTRY FIELDS CANNOT BE EMPTY !!
Okay

- T7E2:

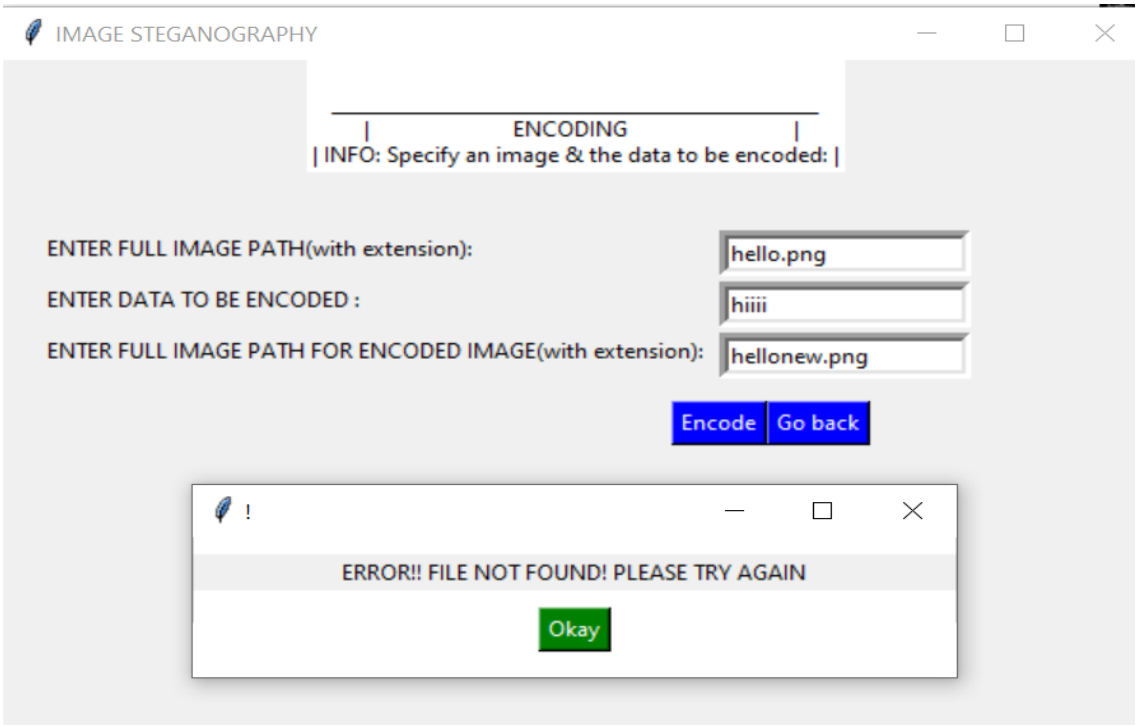
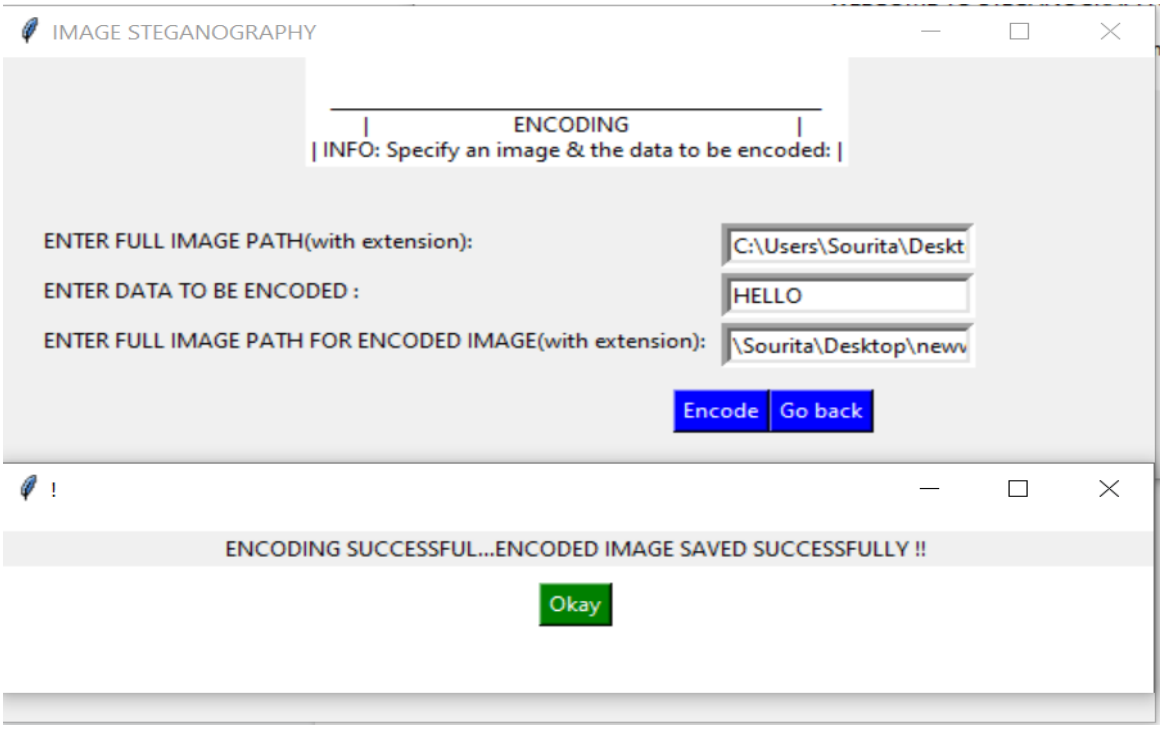
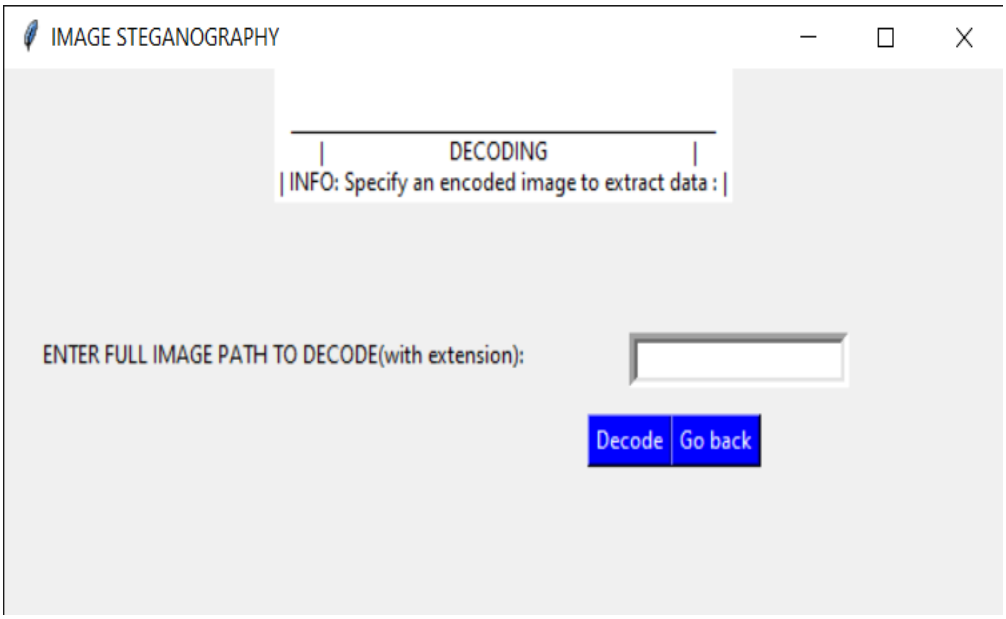


IMAGE STEGANOGRAPHY
ENCODING
| INFO: Specify an image & the data to be encoded: |
ENTER FULL IMAGE PATH(with extension):
hello.png
ENTER DATA TO BE ENCODED :
hiiii
ENTER FULL IMAGE PATH FOR ENCODED IMAGE(with extension):
hellonew.png
Encode
Go back
!
ERROR!! FILE NOT FOUND! PLEASE TRY AGAIN
Okay

- T8S1:



4.3 DECODING WINDOW TEST:

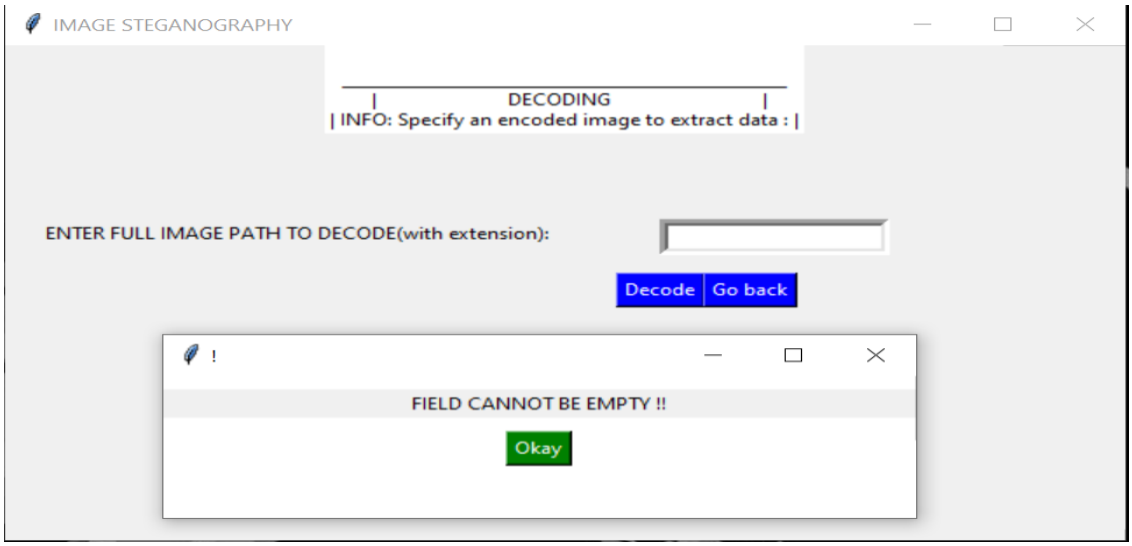


Test Results:

TEST	TEST	EXPECTED RESULT	OUTCOME

ID	DESCRIPTION		
T1	TEXTFIELD FOR IMAGE PATH THAT IS TO BE DECODED	ACCEPT USER INPUT FOR IMAGE PATH.	PASS
T2	DECODE BUTTON	WHEN PRESSED THE GIVEN IMAGE SHOULD BE PROCESSED FOR DECODING.	PASS
T3	GO BACK BUTTON	WHEN PRESSED USER SHOULD BE BROUGHT BACK TO THE MAIN WINDOW.	PASS
T4E1	ERROR POPUP-1	SHOULD POPUP WHEN THE USER LEAVES ANY OF THE ABOVE TEXT FIELDS EMPTY.	PASS
T5E2	ERROR POPUP-2	SHOULD POPUP ERROR MESSAGE WHEN THE GIVEN IMAGE PATH IS NOT VALID.	PASS
T6S1	SUCCESSFUL POPUP	SHOULD POPUP A MESSAGE BOX WHEN ALL THE PROCESS HAS BEEN COMPLETED SUCCESSFULLY SHOWING THE DECODED DATA.	PASS

- T4E1:



- T5E2:

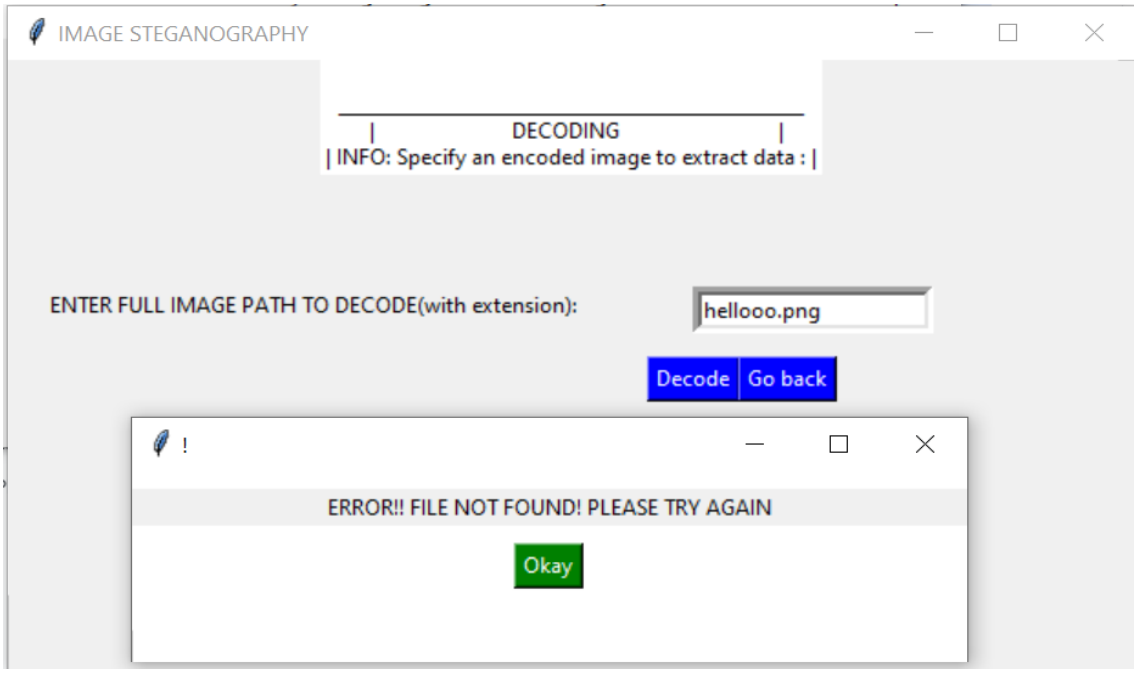


IMAGE STEGANOGRAPHY
DECODING
| INFO: Specify an encoded image to extract data : |
ENTER FULL IMAGE PATH TO DECODE(with extension):
hellooo.png
Decode
Go back
!
ERROR!! FILE NOT FOUND! PLEASE TRY AGAIN
Okay

- T6S1:

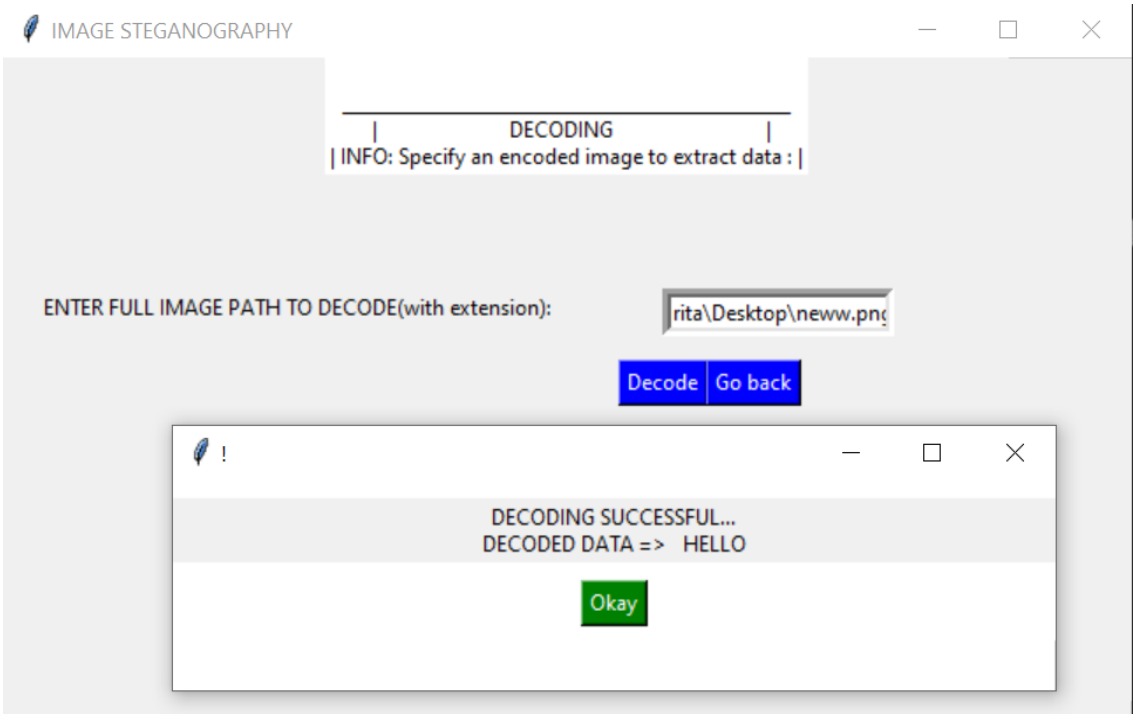
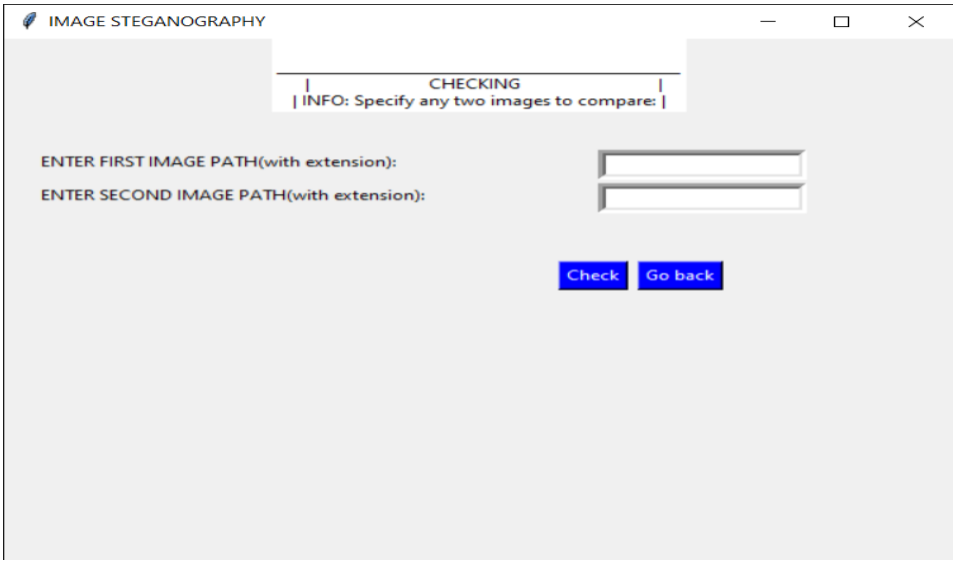


IMAGE STEGANOGRAPHY
DECODING
| INFO: Specify an encoded image to extract data : |
ENTER FULL IMAGE PATH TO DECODE(with extension):
rita\Desktop\neww.pn
Decode
Go back
!
DECODING SUCCESSFUL...
DECODED DATA => HELLO
Okay

4.4 CHECKING WINDOW TESTING:



Test Results:

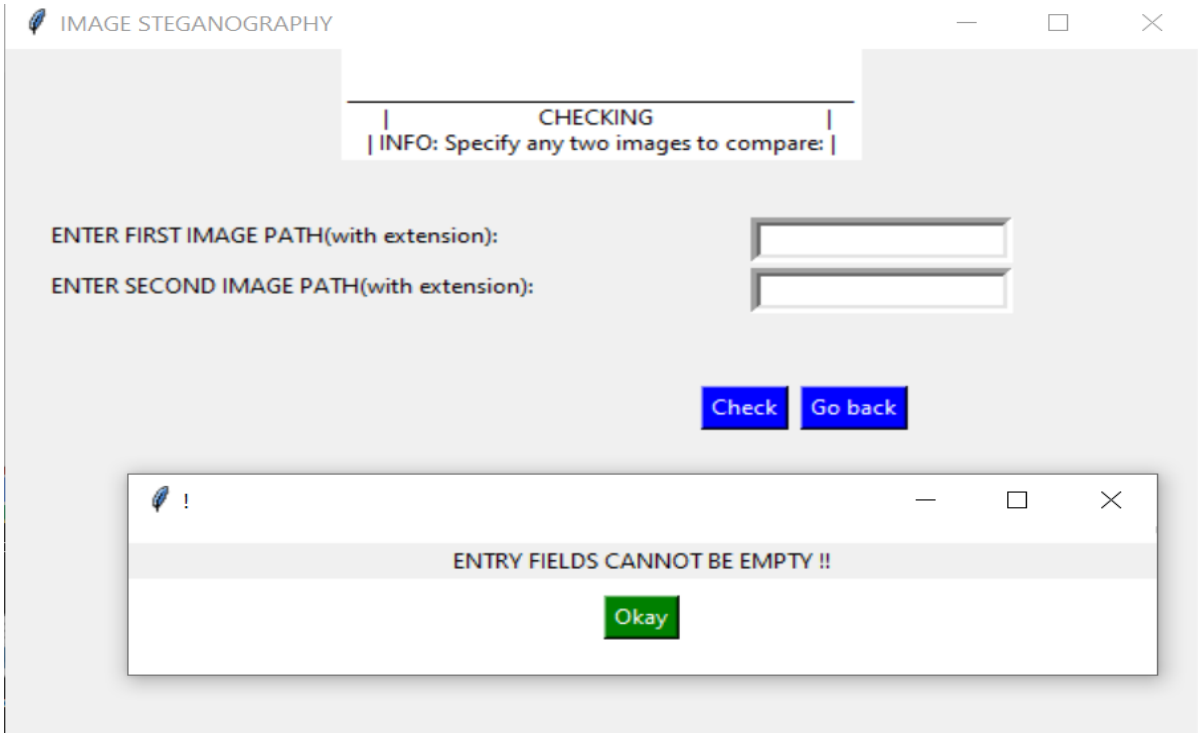
TEST ID	TEST DESCRIPTION	EXPECTED RESULT	OUTCOME
T1	TEXTFIELD FOR FIRST IMAGE PATH THAT IS TO BE COMPARED.	ACCEPT USER INPUT FOR IMAGE PATH.	PASS
T2	TEXTFIELD FOR SECOND IMAGE PATH FOR COMPARISON.	ACCEPT USER INPUT FOR IMAGE PATH.	PASS
T3	CHECK BUTTON	WHEN PRESSED THE GIVEN IMAGES SHOULD BE PROCESSED FOR SIMILARITIES OR DIFFERENCES.	PASS
T4	GO BACK BUTTON	WHEN PRESSED USER SHOULD BE BROUGHT BACK TO THE MAIN WINDOW.	PASS
T5E1	ERROR POPUP-1	SHOULD POPUP WHEN THE USER LEAVES ANY OF THE ABOVE TEXT FIELDS EMPTY.	PASS
T5E1	ERROR POPUP-2	SHOULD POPUP ERROR	PASS

		MESSAGE WHEN THE GIVEN IMAGE PATHS ARE NOT VALID.	
T7S1	SUCCESSFUL POPUP-1	SHOULD POPUP A MESSAGE BOX WHEN BOTH IMAGES ARE SIMILAR.	PASS
T8S2	SUCCESSFUL POPUP-2	SHOULD POPUP A MESSAGE BOX WHEN BOTH IMAGES ARE NOT SIMILAR.	PASS

NOTE:

- Below are the attached snapshots to validate the tests done.

  - T5E1:



- T6E2:

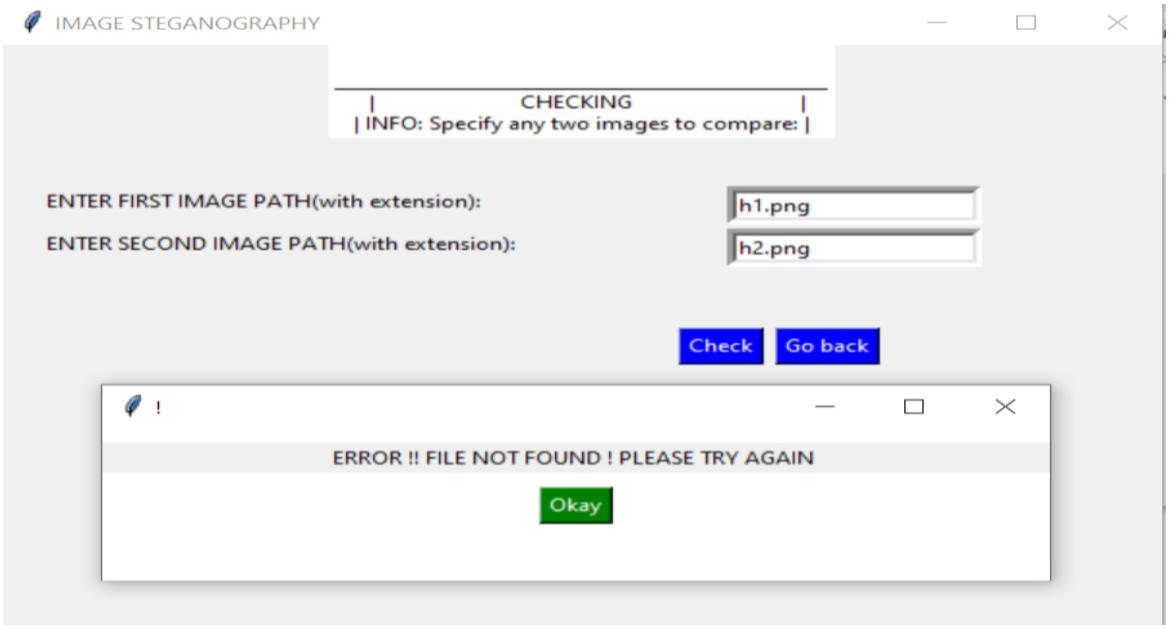


IMAGE STEGANOGRAPHY
CHECKING
INFO: Specify any two images to compare:
ENTER FIRST IMAGE PATH(with extension):
h1.png
ENTER SECOND IMAGE PATH(with extension):
h2.png
Check
Go back
!
ERROR !! FILE NOT FOUND ! PLEASE TRY AGAIN
Okay

- T7S1:

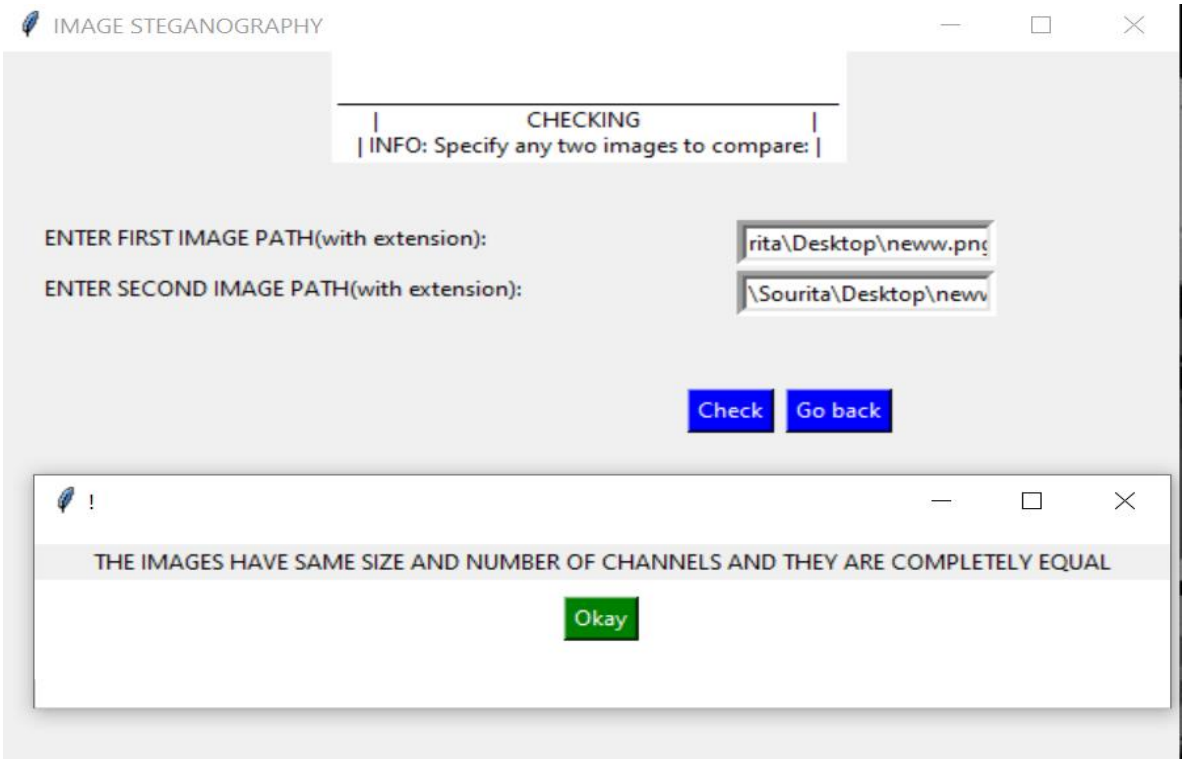
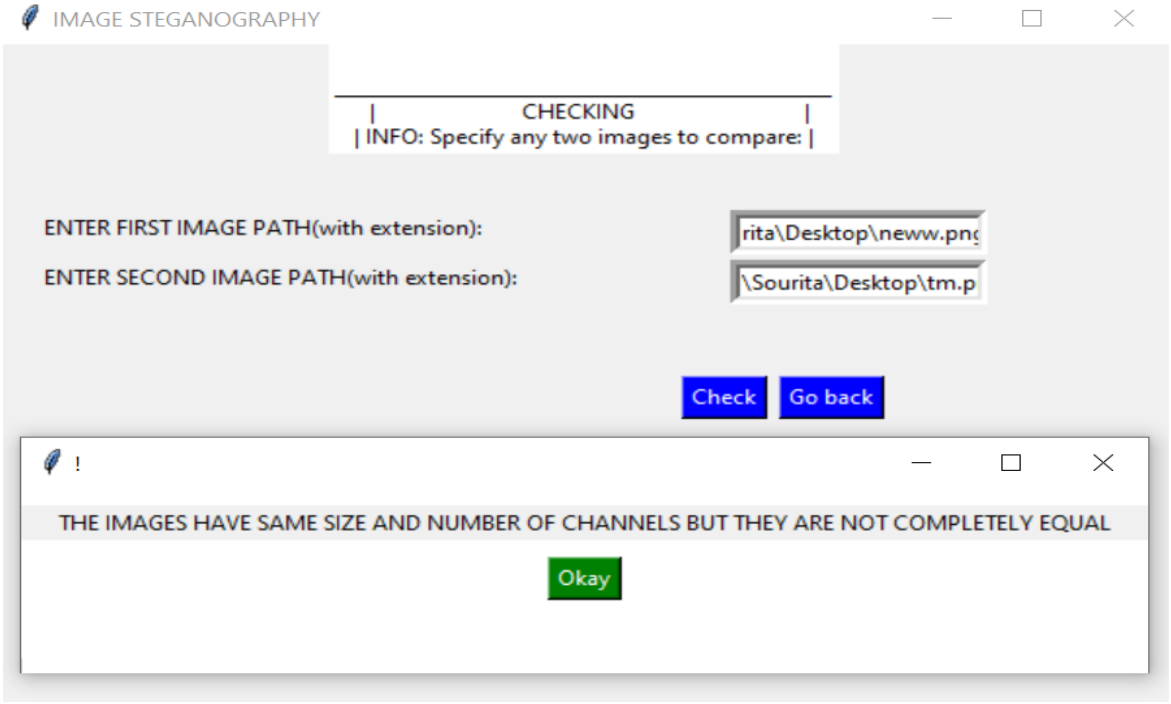


IMAGE STEGANOGRAPHY
CHECKING
INFO: Specify any two images to compare:
ENTER FIRST IMAGE PATH(with extension):
rita\Desktop\neww.png
ENTER SECOND IMAGE PATH(with extension):
\Sourita\Desktop\neww
Check
Go back
!
THE IMAGES HAVE SAME SIZE AND NUMBER OF CHANNELS AND THEY ARE COMPLETELY EQUAL
Okay

- T8S2:



The overall testing performed on the system has met the expected result. The system seems to be stable and reliable as the system did not crash throughout tests. The system successfully encodes, decodes and checks the respective digital images. The feedback of testing was very positive. Therefore, the system is now ready for evaluation against the full run.
