BIOMETRIC STEGANOGRAPHY

MAJOR PROJECT REPORT

Submitted in fulfilment of the requirements for the award of the degree of

M. Sc. Computer Science

Submitted by

Anu (1893894)

Anuradha Aggarwal (1893895)

Under the Supervision of

Prof. Sunil Kumar Muttoo

Department of Computer Science, Faculty of Mathematical Sciences, University of Delhi, New Delhi, India

2020

ACKNOWLEDGEMENT

It was an intellectually enriching experience to undertake this major project as a part of our MSc. Computer science 4th semester curriculum at Department of Computer Science, University of Delhi.

We sincerely express our gratitude towards our project supervisor **Prof. S.K. Muttoo**, Department of Computer Science, University of Delhi, who guided us at every step in the conceptual undertaking and the implementation of the project work.

We also express our thanks to the Head of Department of Computer Science**, Prof. Neelima Gupta**, for providing us the facilities to carry out the project work.

We also acknowledge the cooperation received from the entire laboratory and office staffs.

DECLARATION

We hereby declare that the project work entitled **Biometric Steganography** being submitted in fulfilment of the requirements for the award of the degree of M.Sc. (Computer Science), is a record of original and bona-fide work carried out by the undersigned in the Department of Computer Science, Faculty of Mathematical Sciences, University of Delhi, New Delhi, India. The work presented in this Project has not been submitted to any other Institute or University for the award of any degree or diploma.

-------------------------	-------------------------
Anu	**Anuradha Aggarwal**

TABLE OF CONTENTS

Abstract

This report presents a new framework for secure data hiding by combining steganography technique with biometrics traits. Steganography is the art and science of hiding the secret data in to the cover file in such a way that no one apart from sender and intended recipient, suspects the secret message. Here image steganography technique is used, in which secret data is embedded in the image file which acts like cover file. Traditional steganographic techniques uses sequential LSB data embedding method, in which secret data is going to embed in the least significant bit position of cover image pixels in sequential manner, this method is easily vulnerable to the attacks and attacker can easily get know to the data embedded position because data is embedded in sequential pixel position in least significant bits. Hence to overcome from this problem our project proposes new framework called "Biometric steganography". In this steganography is implemented by utilizing two of the biometric traits i.e. Hand geometry & Face geometry. Here features are extracted from hand images & face images of individuals and by utilizing these features one unique key is generated. This generated key is employed at two places. First it is used to encrypt the user data using various cryptography techniques. Secondly, key is used to find a particular pixel position from which, to embed the secret data in least significant bits of the cover image. Stego image is generated which contains secret data, and embedded secret message is extracted from the stego image by utilizing generated unique key, during information extraction. This approach gives a double layered data security.

Introduction

Nowadays the security has become one of the essential issues in the case of secure data transmission. We should take care that the transmission of a data won't get altered by any kind of network attacks and secure transmission of data won't susceptible to any kind of malicious attacks while the data is in transmission mode.

Steganography techniques are used to conceal the secret data and to protect the confidential data. Steganography is the art of invisible communication by concealing information inside other information. The term steganography is derived from Greek word which means "covered writing". Steganography plays an important role in information security. The goal of Steganography is to avoid drawing suspicion to the existence of a hidden message.

This report presents new framework called "Biometric steganography" which gives double layered security for data which has been embedded in the cover file. "Biometric Steganography" is an art of hiding the data in to another data by using some of the biometric traits". Biometrics is metrics related to the human characteristics and it automatically recognizes individuals based on "physical or behavioral" characteristics.

In this frame work steganography technique is used in combination with biometric traits. Steganography is the technique of concealing the secret data in to the one of the cover files, so that no one can suspect the data apart from the sender and receiver.

Here image steganography strategy has been used in which the secret data is going to embed in the cover image. In this proposed system new embedding technique is implemented in which secret data is secretly embedded in cover image based on key, unique key is generated by using extracted hand and face features and this key identifies particular pixel of cover image in which data is going to embed. In hand geometry first hand image has been taken out from database and taken image is preprocessed by cropping unwanted part of the hand image after this process by utilizing the boundary tracing algorithm hand features being extracted and tip point are located using Euclidean distance, by using these extracted features a unique key has generated. In face geometry, first face image has been captured and after preprocessing the image, face and eyes coordinates has been detected and these coordinates is used to generate a unique key. This key identifies a particular pixel of cover image in which secret data embeds in the least significant bits position of that selected pixel. Stego image is generated which contains secret data within it, from this stego image data being extracted by performing reverse process of embedding, secret data is extracted using the same hand key which has been used while embedding.

Chapter 1: Steganography

Types of steganography

There are broadly **four types** of Stenography which are as follows:

Text Steganography: General technique in text steganography, such as number of tabs, white spaces, capital letters, just like Morse code (and etc.) is used to achieve information hiding.

Text Steganography Techniques:

a) Selective hiding: This hides the characters in the first (or any specific location) characters of the words. Concatenating those characters help extracting the text. But this technique requires huge amount of plain text.
b) HTML web pages: This may hide text using the fact that attributes of HTML tags are case insensitive. Those characters can then be used to retrieve the original text.
c) Hiding using Whitespace: Fewer numbers of whitespaces may specify a 0 and more number of whitespaces between words may determine a 1.
d) Semantic Hiding: Uses synonyms to hide the message.

Video Steganography: Video Steganography is a technique to hide any kind of files or information into digital video format. Video (combination of pictures) is used as carrier for hidden information. Generally discrete cosine transform (DCT) alter values (e.g., 8.667 to 9) which is used to hide the information in each of the images in the video.

Audio Steganography: When taking audio as a carrier for information hiding it is called audio steganography. It has become very significant medium due to voice over IP (VOIP) popularity. Audio steganography uses digital audio formats such as WAVE, MIDI, AVI MPEG or etc for steganography.

Image Steganography: Taking the cover object as image in steganography is known as image steganography. Generally, in this technique pixel intensities are used to hide the information.

Image Steganography Techniques

The following generic embedding techniques can be found in the literature, particularly for image steganography:

a) Least Significant Bit (LSB) Embedding

Each pixel of a red–green–blue (RGB) image is represented by 24 bits, which is an 8-bit binary string covering decimal values 0 to 255 for each of the three red, green, and blue channels. The least significant bit (LSB) of one of these strings is the last (or right-most) binary integer that gives the unit value .

Deliberate alteration of the LSB, or indeed the last two binary digits, can be used to embed secret information into that pixel without the change being detectable to the human eye viewing the image .

LSB-Steganography is a steganography technique in which we hide messages inside an image by replacing Least significant bit of image with the bits of message to be hidden.

By modifying only the first most right bit of an image we can insert our secret message and it also makes the picture unnoticeable, but if our message is too large it will start modifying the second rightmost bit and so on and an attacker can notice the changes in picture.

Advantages:

1. Easy implementation.
2. Fast.
3. High capacity when using 4-LSB embedding.

Disadvantages:

1. Vulnerable to steganalysis attacks in the spatial domain.
2. Loss of image quality with greater than three bits embedded per pixel.

b) Discrete Cosine Transform (DCT) Embedding

Discrete cosine transform (DCT) is a mathematical transformation which takes an image block in a spatial domain and transforms it into a frequency domain consisting of high, medium, and low-frequency components or sub-bands.

Once the embedding is complete, an inverse DCT algorithm is applied to transform the signal coefficients back to the spatial domain.

Advantages:

1. Easy implementation.
2. Robust against cropping and compression

Disadvantages:

1. Lower embedding capacity.
2. Poor quality.
3. Low security.

c) Discrete Wavelet Transform (DWT)

Discrete wavelet transform (DWT) is a mathematical transformation which takes an image’s wavelet in the spatial domain and transforms it into the frequency domain. However, the main difference between DWT and DCT is in the high-pass bands. DWT provides lower frequency resolution, but higher spatial resolution. It, therefore, contains fewer sub-bands compared to DCT but has improved spatial resolution.

Advantages:

1. Highly secure.
2. Robust against cropping and compression

Disadvantages:

1. Complex technique requiring considerable computational resources.
2. Only moderate embedding capacity.
3. Requires considerable auxiliary data to be reversible.

Chapter 2: Biometric Steganography

In this proposed system new embedding technique is implemented in which secret data is secretly embedded in cover image based on key, unique key is generated by using extracted hand and face features and this key identifies particular pixel of cover image in which data is going to embed. In hand geometry, first hand image has been taken out from database and taken image is preprocessed by cropping unwanted part of the hand image after this process by utilizing the boundary tracing algorithm hand features being extracted and tip point are located using Euclidean distance, by using these extracted features a unique key has generated. In face geometry, first face image has been captured and after preprocessing the image, face and eyes coordinates has been detected and these coordinates is used to generate a unique key. This key identifies a **particular pixel** of cover image in which secret data embeds in the least significant bits position of that selected pixel. Stego image is generated which contains secret data within it, from this stego image data being extracted by performing reverse process embedding, secret data is extracted using the same hand key which has been used while embedding.

Chapter 3: Implementation

In this project i.e. “Biometric steganography”, steganography technique is implemented along with using two of the biometric traits i.e. hand geometry & face geometry.

Hence implementation part is divided in two parts:

- Hand geometry
- Face geometry

Tools

Language used : python

Platform used : Anaconda

IDE : Jupyter notebook

Packages used:

- Numpy
- PIL
- Cv2
- Matplotlib
- Skimage
- Imutils

Chapter 3a) Hand Biometrics

Phases

This is divided into **8** phases:

1. Hand Feature Extraction
2. Key generation
3. Message Encryption
4. Data Embedding
5. Stego-image Encryption
6. Stego-image Decryption
7. Data Extraction
8. Message Decryption

1.Hand Feature Extraction

Here hand geometric features are extracted by following 3 steps.

- Image Acquisition
- Image preprocessing
- Feature Extraction

1. **Image Acquisition:**

   We need a hand Image for further feature extraction. In this image acquisition phase system consist of “CCD digital camera”, “light source”, “black flat surface”. User normally places his hand on black flat surface and image is captured by the CCD digital camera.

2. **Image Preprocessing:**

   We have acquired the hand image which is colored image; we have to convert it into the black and white image and unnecessary part of the captured hand image has cropped. “Median filter” had used to remove the background noise of the image. There is specific intensity between hand and background which we can observe because of the black colored background. Thus histogram of the hand image is “bimodal”. The hand image can easily changed over to the “binary image” by thresholding. The threshold value consequently figured by utilizing “Otsu” method. Hand image outline can be smoothened by applying some of the morphological operations.

3. **Feature Extraction:**

   After preprocessing stage the feature extraction of hand has performed by using boundary tracing algorithm(trace contours of image used to plot all the contours point and find convex hull) and some of the calculations are used to find Tip points. Euclidean distance is calculated between tip points and center point of hand image.

2.Key generation

Key has generated in four ways -

- From extracted features tip and valley points are located
- Pairwise distance has calculated using Euclidean distance.
- Variance has calculated for pairwise distance
- And lastly unique key has generated using variance result.

Pairwise distance calculated using Euclidean distance

Given an m-by-n data matrix X, which is treated as m (1-by-n) row vectors x_1, x_2... x_m, the various distances between the vector x_r and x_s are defined as follows:

$$d_{rs}^2 = (x_r - x_s)(x_r - x_s)'$$

Variance

The mathematical formula to calculate the variance is given by:

$$\sigma^2 = \frac{\sum (X-\mu)^2}{N}$$

σ^2 = variance

$\sum (X - \mu)^2$ = The sum of $(X - \mu)^2$ for all data points

Standard Deviation Formula

The standard deviation formula is similar to the variance formula. It is given by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

σ = standard deviation

x_i = each value of dataset

x (with a bar over it) = the arithmetic mean of the data (This symbol will be indicated as mean from now)

N = the total number of data points

$\sum (x_i - \text{mean})^2$ = The sum of $(x_i - \text{mean})^2$ for all data points.

Unique 4 digit key generation

Key= (Most Significant 2 Digits of p1) * 10 + (Least Significant 1 Digits p2)

Where

P1 = variance of Hand points pairs wise distance

P2 = standard deviation of Hand points pairs wise distance

- From this unique key we will get unique pixel position to embed the secret data instead of embedding the data in sequential pixels of cover image.
- This technique gives more security and it’s not possible for opponents to find the accurate pixel position where the secret message has been embedded.

3.Message Encryption

In this phase user enters a secret message he/she wants to hide.
Now we apply a “Caeser Cipher” cryptography technique to encrypt the original message into secret message.

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It’s simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

$$\mathbf{E(x) = (x + n)\ mod26}$$

(Encryption phase with shift n)

Thus, to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down. Here we use hand key as a shift (n).

4.Data Embedding

Steps for LSB data insertion-

- First carrier image or cover image has been read and converted in to array of bits.
- Then the secret message which is in the form of bytes/characters are converted into the “ASCII” values and then ASCII values are converted into an array of bits.
- A unique key which has generated from hand geometric features based on this we start embedding the secret data
- Here key will gives unique pixel position of cover image to embed the data least significant bit of particular image pixel of cover object. Here one pixel is equal to one byte.
- Stego-image has been generated which contains secret message embedded within cover image.

5.Stego-image Encryption

After stego-image is generated we encrypt this stego-image using AES Cryptography(CBC-mode) technique which gives extra layer of security.

The Advanced Encryption Standard, also known by its original name Rijndael, is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology in 2001. AES is a symmetric, block cipher which means that blocks of text of a certain size (128 bits) are encrypted, as opposed to a stream cipher where each character is encrypted one at a time. The symmetric part refers to that the identical key is used for the encryption process, as well as to decrypt the message.

6.Stego-image Decryption

From this point our decryption phase starts. Decrypt the encrypted stego-image using AES (CBC- mode) using 128 bits key. Here key which we use is same as our previous generated key but this key is transformed to 128 bits.

7.Data Extraction

Steps for LSB data extraction-

- The extracting of the embedded data is done in opposite direction of hiding process.
- Here embedded secret data has extracted from stego-image
- While extracting secret data the hand key is used. while embedding the data we have used key to select the embedding position in cover image so while extracting also we need that key to extract secret data
- This gives complete security to the embedded data in the cover image.

8. Message Decryption

After performing all the above steps we get the encrypted msg. To retrieve the original user message we perform Caeser Cipher Decryption technique.

$$\mathbf{D(x) = (x-n) \bmod 26}$$

(Decryption phase with shift n)

Chapter 3b) Face Biometrics

Phases

This is divided into 6 phase:

1. Face Feature Extraction
2. Key Generation
3. Message Encryption
4. Data Embedding
5. Data Extraction
6. Message Decryption

1.Face Feature Extraction

For face and eyes recognition we use Haar Cascade Algorithm.

Haar Cascade Algorithm:-

It is a machine learning algorithm used to identify objects in image or video based on the concepts of features proposed by Paul Viola & Michael Jones.

It is a classification which classifies between a face or non-face image. which are trained from many positive images (with faces) and negatives images (without faces).

The algorithm contains four stages:

- Haar Feature Selection
- Creating Intergal Images
- Adaboost Training
- Cascading Classifiers

2.Key Generation

From above algorithm we get the coordinates of face, left eye & right eye.
We use these coordinates to compute the distances between:-

1.Left eye and face
2.Right eye and face
3.Left eye and right eye

Now compute these distances to generate unique key for our next steps.

Key = (Most Significant 2 Digits of p1) * 10 + (Least Significant 1 Digits p2)

Where,

P1 = variance of Hand points pairwise distance

P2 = standard deviation of Hand points pairwise distance

3.Message Encryption

In this phase user enters a secret message he/she wants to hide.
Now we apply a “Transposition Cipher” cryptography technique to encrypt the original message into secret message.

In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext.

Transposition Cipher Encryption

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.

2. Width of the rows and the permutation of the columns are usually defined by a keyword.

3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.

4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).

5. Finally, the message is read off in columns, in the order specified by the keyword.

4.Data Embedding

Steps for LSB data insertion-

- First carrier image or cover image has been read and converted in to array of bits.
- Then the secret message which is in the form of bytes/characters are converted into the “ASCII” values and then ASCII values are converted into an array of bits.
- A unique key which has generated from face geometric features based on this we start embedding the secret data
- Here key will gives unique pixel position of cover image to embed the data least significant bit of particular image pixel of cover object. Here one pixel is equal to one byte.
- Stego-image has been generated which contains secret message embedded within cover image.

5.Data Extraction

Steps for LSB data extraction-

- The extracting of the embedded data is done in opposite direction of hiding process.
- Here embedded secret data has extracted from stego-image
- While extracting secret data the face key is used. while embedding the data we have used key to select the embedding position in cover image so while extracting also we need that key to extract secret data
- This gives complete security to the embedded data in the cover image.

6.Message Decryption

After performing all the above steps we get the encrypted msg. To retrieve the original user message we perform Transposition Cipher Decryption technique.

Decryption

1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.

2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

Chapter 4: Execution

Chapter 4a): Hand Biometric Execution

Step1: Image Preprocessing

```
'''
 --Acquired the coloured hand image
 --resize the image
 --rotate the image by -90-degree

'''
originalImage = cv2.imread('hand.jpg')
resizedImage1 = cv2.resize(originalImage, (200,200))
resizedImage = imutils.rotate(resizedImage1, -90)
displayImage("resized",resizedImage)
```

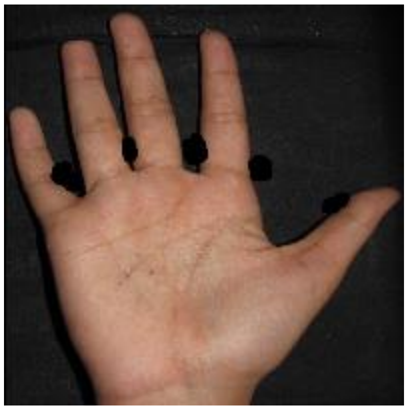


fig 1: Original and resized hand image

''' convert the hand-image into Gray image '''

```
#convert the image into GrayImage

grayImage = cv2.cvtColor(resizedImage, cv2.COLOR_BGR2GRAY)

#convert the image into black and white

(thresh, blackAndWhiteImage) = cv2.threshold(grayImage, 127, 255,
cv2.THRESH_BINARY)

displayImage('Black white image', blackAndWhiteImage)

displayImage('Original image',resizedImage)

displayImage('Gray image', grayImage)
```

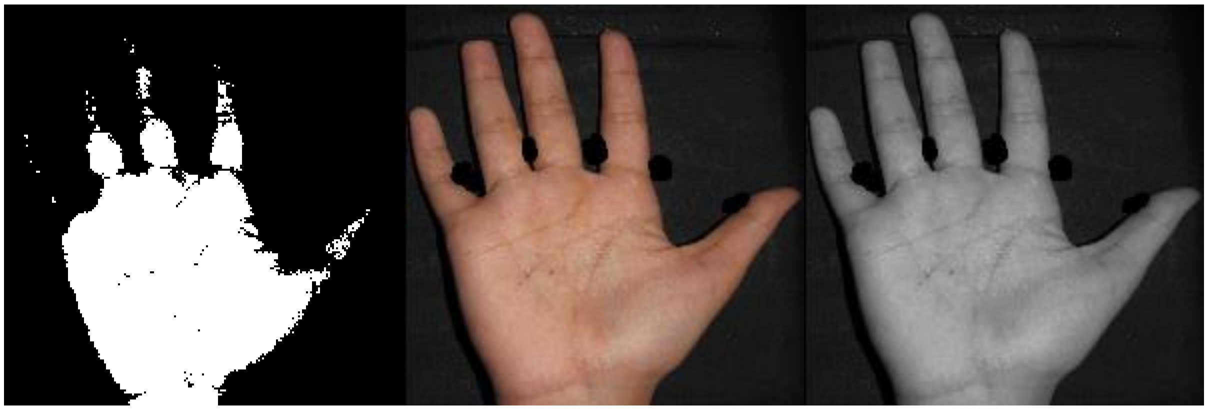


fig 2: Black and white hand image, Blurred resized hand image, Gray hand image

```
'''Gaussian filter had used to remove the background noise of the image '''

#gaussian filtering

blur = cv2.GaussianBlur(grayImage,(5,5),0)

displayImage("Gaussianblur",blur)
```

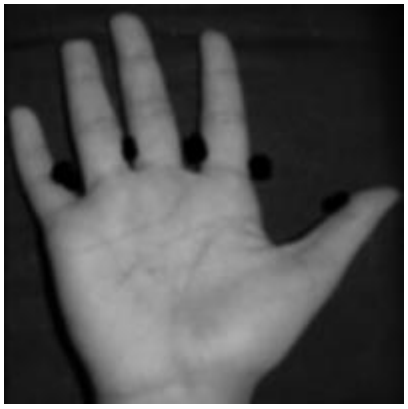


fig 3: Image after applying Gaussian filter

```
'''

hand image is changed over to the "binary image" by thresholding

# applying Otsu thresholding

# as an extra flag in binary

# thresholding

'''

ret, thresh1 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

displayImage('Otsu Threshold', thresh1)

ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

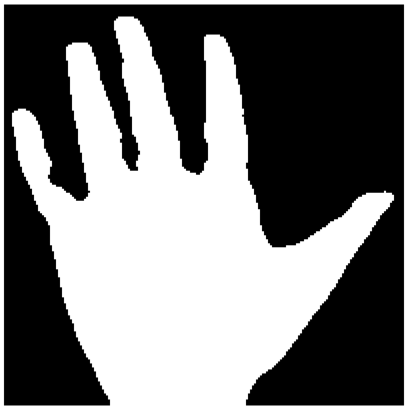


fig 4: Binary Image by applying Otsu Thresholding

Step 2: Feature Extraction

```
'''
Boundary Tracing of the preprocessed hand image
-trace contours of image used to plot all the contours point
'''
cnts = cv2.findContours(thresh1, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
#print(cnts)
cnts = imutils.grab_contours(cnts)
c = max(cnts, key=cv2.contourArea)
# determine the most extreme points along the contour
extLeft = tuple(c[c[:, :, 0].argmin()][0])
extRight = tuple(c[c[:, :, 0].argmax()][0])
extTop = tuple(c[c[:, :, 1].argmin()][0])
extBot = tuple(c[c[:, :, 1].argmax()][0])

# draw the outline of the object, then draw each of the
# extreme points, where the left-most is red, right-most
# is green, top-most is blue, and bottom-most is teal
cv2.drawContours(thresh1, [c], -1, (125, 125, 125), 2)
cv2.circle(thresh1, extLeft, 8, (50, 50, 50), -1)
cv2.circle(thresh1, extRight, 8, (50, 50, 50), -1)
cv2.circle(thresh1, extTop, 8, (50, 50, 50), -1)
```

```
cv2.circle(thresh1, extBot, 8, (50, 50, 50), -1)

displayImage("Image", thresh1)
```

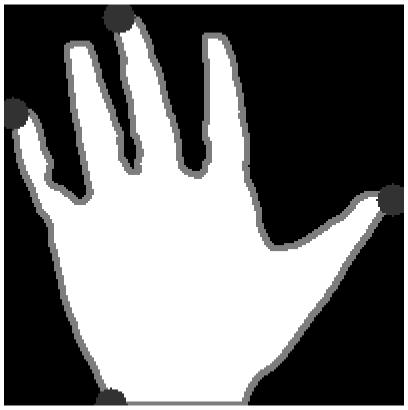


fig 5: Most extreme points along the contour of hand image showing extreme left ,right and top down points

```
#print all the contours point

items = [tuple(item[0]) for npar in cnts for item in npar]

#print(items)

xRes = [lis[0] for lis in items]

yRes = [lis[1] for lis in items]

plt.plot(yRes,xRes)
```

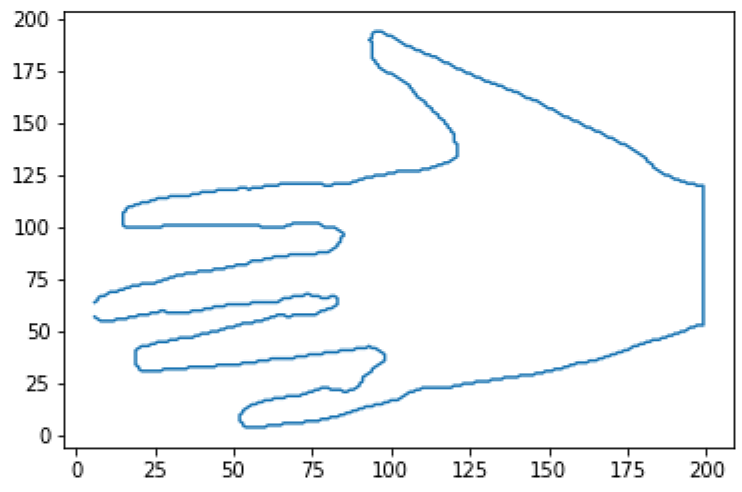


fig 6: Plot to show Boundary Tracing of the preprocessed hand image

```
'''

- Detect the convex contour

- find the X-Y coordinates of hand image

- find convex hull

'''

hull = cv2.convexHull(cnts[0])

#print(hull)

xhull = [item[0] for sublist in hull for item in sublist]

yhull = [item[1] for sublist in hull for item in sublist]

#print('convexHull-X-coord:= ',xhull)

#print('convexHull-Y-coord:= ',yhull)

img_copy = thresh1.copy()

img_hull = cv2.drawContours(img_copy, contours = [hull], contourIdx = 0, color = (125,
125, 125), thickness = 2)

plt.imshow(img_hull)
```

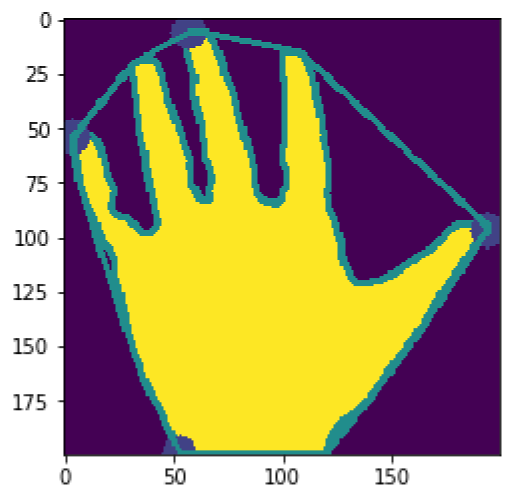


fig 7: Plot to show convex hull of hand image

```
'''

- find the coordinates of center point for hand image

'''

# calculate moments of binary image

M = cv2.moments(th3)

# calculate x,y coordinate of center

cX = int(M["m10"] / M["m00"])

cY = int(M["m01"] / M["m00"])

#print("cX",cX)

#print("cY",cY)

# put text and highlight the center

cv2.circle(th3, (cX, cY), 5, (125, 125, 255), -1)

cv2.putText(th3, "centroid", (cX - 25, cY - 25),cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255, 255, 255), 2)

# display the image
```

```
displayImage("Image", th3)
```

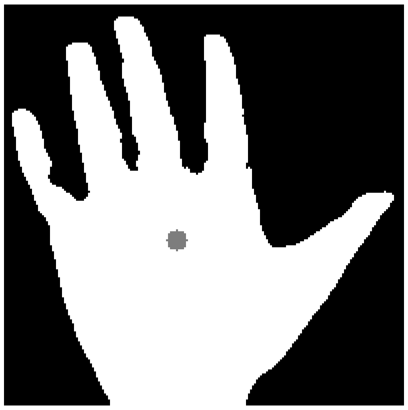


fig 8: Preprocessed image to highlight the center point of hand image

```
'''
- plot the X-Y coordinates of contours point and center point
'''

plt.plot(yRes,xRes)
plt.plot(yhull,xhull,'ro')
plt.plot(cY,cX,'go')
plt.xlabel("yhull")
plt.ylabel("xhull")
```

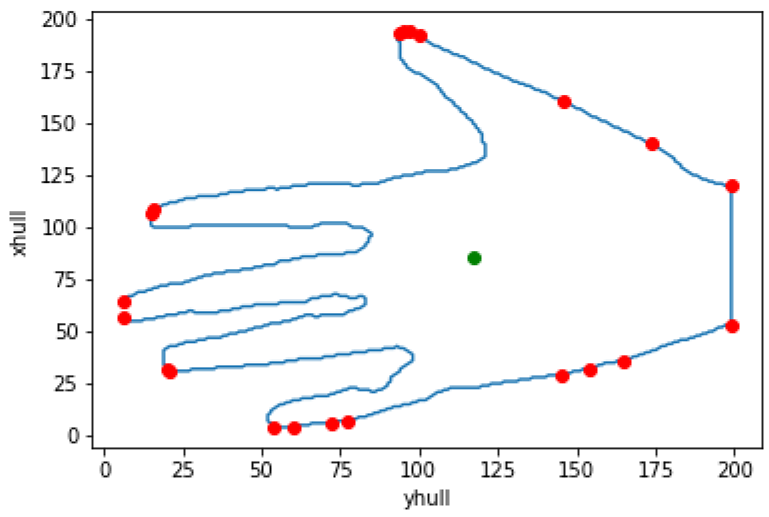


fig 9: Plot to show the X-Y coordinates of contours point and center point of hand image

```
'''
 To find the tips point from hand geometry
 Consider the points above the center point which indicate the tip points
'''


#only take the coordinates of tip points
xhullUpdated = []
yhullUpdated = []
t=0;
for i in yhull:
    if(i<cY):
        yhullUpdated.append(i)
        xhullUpdated.append(xhull[t])
    t=t+1
```

```
plt.plot(yRes,xRes)

plt.plot(yhullUpdated,xhullUpdated,'ro')

plt.plot(cY,cX,'go')

plt.xlabel("yhullUpdated")

plt.ylabel("xhullupdated")
```

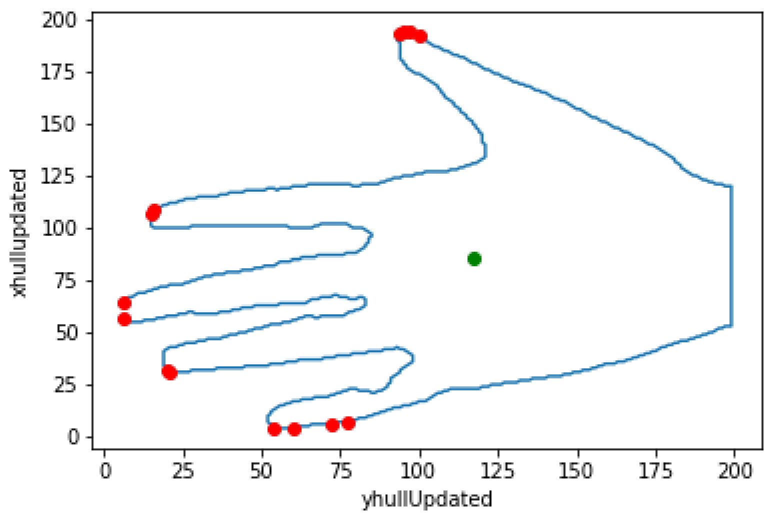


fig 10: Plot to display tips point from hand geometry and center point

Step 3: Key Generation

```
'''

Euclidean distance between center point and all other points of convex hull(tips point)

'''

distVector = []

length=int(len(yhullUpdated))

for i in range(length):

    distVector.append(np.sqrt( (xhullUpdated[i]-cX)**2 + (yhullUpdated[i]-cY)**2 ))
```

```
'''
Variance and standard deviation is calculated for pairwise Euclidean distance
'''
import statistics


variance = statistics.variance(distVector)
std_dev = statistics.stdev(distVector)
print("Variance of sample set is % s"  %(variance))
print("Standard Deviation of sample is % s " %(std_dev))
Output:-  Variance of sample set is 59.81944190387565
Standard Deviation of sample is 7.734302935874418


'''
Unique 2 digit key generation
Key= (Most Significant 2 Digits of p1) * 100 + (Least Significant 1 Digits p2)
Where
P1 = variance of Hand points pairs wise distance
P2 = standard deviation of Hand points pairs wise distance


'''
MSB = int(str(variance)[:2])
LSB = int(str(std_dev)[-1:])
```

```
#print(MSB,LSB)

key = (MSB)* 10 + (LSB)

print("Key Generated:- ",key)
```

```
Output:  Key Generated:-  59
```

Step 4: Message Encryption using Caeser Cipher

```
hand_key = key
Number = hand_key
Sum = 0
while(Number > 0):
    r = Number % 10
    Sum = Sum + r
    Number = Number//10
#print(Sum)

def encrypt(text,s):
    result = ""
    #print("lrngth" , len(text))
    for i in range(len(text)):
        char = text[i]
        # Encrypt uppercase characters in plain text
        if (char.isupper()):
            result += chr((ord(char) + s-65) % 26 + 65)
        # Encrypt lowercase characters in plain text
        else:
            result += chr((ord(char) + s - 97) % 26 + 97)
    return result

text = input("Enter data to be encoded(Without Space) : ")
#print ("Plain Text : " + text)
#print ("Shift pattern : " + str(Sum))
print ("Cipher-Msg: " + encrypt(text,Sum))
```

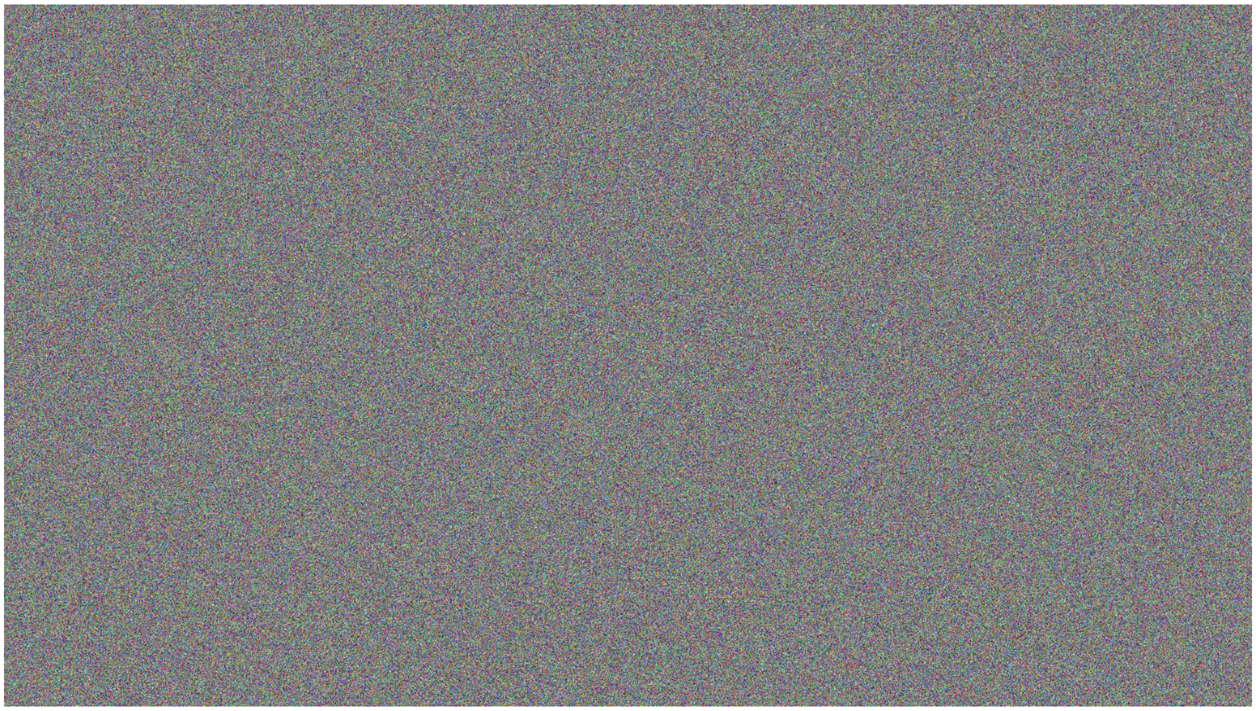
Step 5: Data Embedding

```
def to_bin(data):
    """Convert `data` to binary format as string"""
    if isinstance(data, str):
        return ''.join([ format(ord(i), "08b") for i in data ])
    elif isinstance(data, bytes) or isinstance(data, np.ndarray):
        return [ format(i, "08b") for i in data ]
    elif isinstance(data, int) or isinstance(data, np.uint8):
        return format(data, "08b")
    else:
        raise TypeError("Type not supported.")


'''
- First carrier image or cover image has been read and converted in to array of bits
- the secret message which is in the form of characters are converted in to the "ASCII" values
- ASCII values are converted in to array of bits
- key will gives unique pixel position of cover image to embed the data LSB of particular image pixel of cover object

Output: Stego-image has been generated which contains secret message embedded within cover image.

'''
res = input("Enter data to be encoded(Without Space) : ")
data = ''.join(format(ord(i), 'b') for i in res)
#print("The string after binary conversion : " + data)
msg_len = len(data)
#print("msg length",msg_len)


i=0

with Image.open("source_img.png") as img:

    width, height = img.size
    print(width)
    print(height)
    #img.show()
    for x in range(key, width):
        for y in range(key, height):
            pixel = list(img.getpixel((x, y)))
            for n in range(0,1):
                if(i < len(data)):
                    pixel[n] = pixel[n] & ~1 | int(data[i])
                    i+=1
            img.putpixel((x,y), tuple(pixel))


    img.save("source_secret.png", "PNG")
```

Step 6: AES Cryptography (CBC-mode) Encryption

```
from PIL import Image
from Crypto.Cipher import AES

filename = "source_secret.png"
filename_out = "AES_encrypted_image"
format = "PNG"

#hand_key=5980
key = str(hand_key)*8
#print('key', key)


# AES requires that plaintexts be a multiple of 16, so we have to pad the data
def pad(data):
    return data + b"\x00"*(16-len(data)%16)

# Maps the RGB
def convert_to_RGB(data):
    r, g, b = tuple(map(lambda d: [data[i] for i in range(0,len(data)) if i % 3 == d], [0, 1, 2]))
    pixels = tuple(zip(r,g,b))
    return pixels

def process_image(filename):
    # Opens image and converts it to RGB format for PIL
    im = Image.open(filename)
    data = im.convert("RGB").tobytes()

    # Since we will pad the data to satisfy AES's multiple-of-16 requirement, we will store
the original data length and "unpad" it later.
    original = len(data)

    # Encrypts using desired AES mode (we'll set it to ECB by default)
    new = convert_to_RGB(aes_cbc_encrypt(key, pad(data))[:original])

    # Create a new PIL Image object and save the old image data into the new image.
    im2 = Image.new(im.mode, im.size)
    im2.putdata(new)

    #Save image
    im2.save(filename_out+"."+format, format)


# CBC
def aes_cbc_encrypt(key, data, mode=AES.MODE_CBC):
    IV = "A"*16  #We'll manually set the initialization vector to simplify things
    aes = AES.new(key, mode, IV)
    new_data = aes.encrypt(data)
    return new_data
# ECB
def aes_ecb_encrypt(key, data, mode=AES.MODE_ECB):
    aes = AES.new(key, mode)
    new_data = aes.encrypt(data)
    return new_data

process_image(filename)
```

Output :- Encrypted Stego-image



Step 7: AES Cryptography (CBC-mode) Decryption

```
def aes_cbc_decrypt(key, data, mode=AES.MODE_CBC):
    IV = "A"*16
    aes = AES.new(key, mode, IV)
    decd = aes.decrypt(data)
    return decd


def decrypt_image(filename):
    # Opens image and converts it to RGB format for PIL
    im = Image.open(filename)
    data = im.convert("RGB").tobytes()

    # Since we will pad the data to satisfy AES's multiple-of-16 requirement, we will store
the original data length and "unpad" it later.
    original = len(data)

    # Decrypts using desired AES mode (we'll set it to ECB by default)
    new = convert_to_RGB(aes_cbc_decrypt(key, pad(data))[:original])

    # Create a new PIL Image object and save the old image data into the new image.
    im3 = Image.new(im.mode, im.size)
    im3.putdata(new)

    #Save image
    im3.save(filename_in+"."+format, format)

filenames = filename_out+"."+format
filename_in = "AES_decrypted_image"
decrypt_image(filenames)
```

Output:- Decrypted stego-image

Step 8: Data Extraction

```
'''
- Embedded secret data has extracted from stego-image
- Extracting secret data the hand key is used

'''
extracted_bin = []
with Image.open("source_secret.png") as img:
    width, height = img.size
    byte = []
    for x in range(key, width):
        for y in range(key, height):
            pixel = list(img.getpixel((x, y)))
            for n in range(0,1):
                extracted_bin.append(pixel[n]&1)

data = "".join([str(x) for x in extracted_bin])
msg =''

for i in range(msg_len):
    msg+=str(extracted_bin[i])
#print(msg)
```

```
decoded_msg = ''.join(chr(int(msg[i*7:i*7+7],2)) for i in range(len(msg)//7))
print("decoded_msg:= ",decoded_msg)
```

Step 9: Message Decryption

```
def decrypt(text,s):
    result = ""
    #print("lrngth" , len(text))
    for i in range(len(text)):
        char = text[i]
        # Encrypt uppercase characters in plain text
        if (char.isupper()):
            result += chr((ord(char) - s-65) % 26 + 65)
        # Encrypt lowercase characters in plain text
        else:
            result += chr((ord(char) - s - 97) % 26 + 97)
    return result

print ("Received_msg : " + decoded_msg)
print ("Decipher: " + decrypt(decoded_msg,Sum))
```

Step 10: Compare images

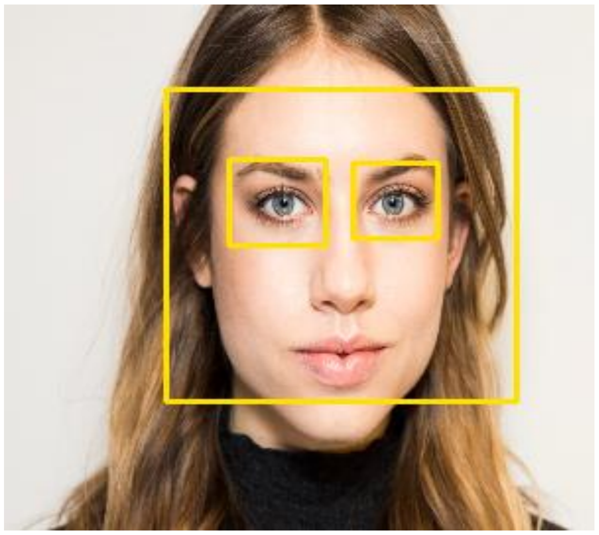
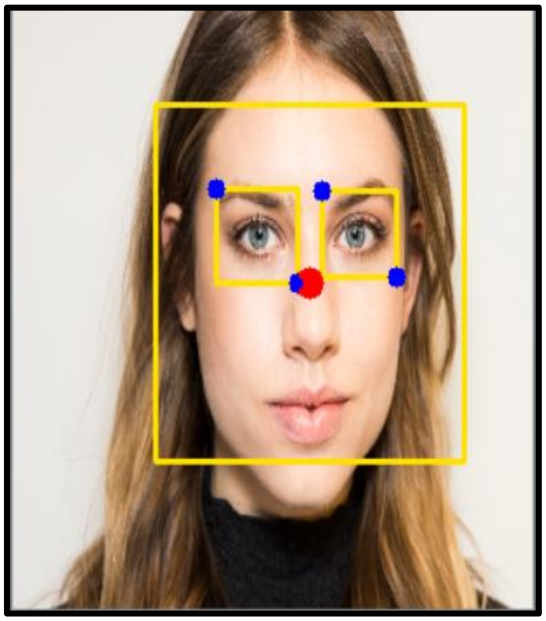
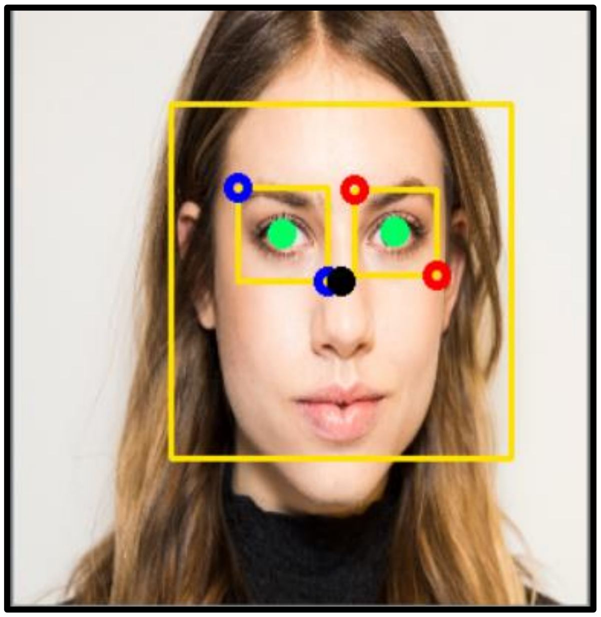
```
#calculate PSNR

import math

original = cv2.imread(r'source_img.png')

updated = cv2.imread(r'source_secret.png',1)

def psnr(img1, img2):

    mse = np.mean( (img1 - img2) ** 2 )

    if mse == 0:

        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
d=psnr(original,updated)
print(d)


#calculate MSE
def mse(original1, contrast1)
    err = np.sum((original1.astype("float") - contrast1.astype("float")) ** 2)
    err /= float(original1.shape[0] * original1.shape[1])
    return err
MSE=mse(original,contrast)
print(MSE)
```

Chapter 4b): Face Biometric Execution

Step 1: Face Feature Extraction

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

originalImage = cv2.imread('face2.png')
img = cv2.resize(originalImage, (300,300))
face = []
leftEyes = []
rightEyes = []

def detect_faces(img, cascade):
    gray_frame = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    coords = cascade.detectMultiScale(gray_frame, 1.3, 5)
    face.append(coords[0][0])
    face.append(coords[0][1])

    print("face",coords)

    if len(coords) > 1:
        biggest = (0, 0, 0, 0)
        for i in coords:
            if i[3] > biggest[3]:
                biggest = i
        biggest = np.array([i], np.int32)
    elif len(coords) == 1:
        biggest = coords
    else:
        return None
    for (x, y, w, h) in biggest:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,225,255),2)

        frame = img[y:y + h, x:x + w]
    return frame
```

```
def detect_eyes(img, cascade):
    gray_frame = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    eyes = cascade.detectMultiScale(gray_frame, 1.3, 5)  # detect eyes
    print("eyes" , eyes)
    leftEyes.append(eyes[0][0])
    leftEyes.append(eyes[0][1])
    rightEyes.append(eyes[1][0])
    rightEyes.append(eyes[1][1])

    width = np.size(img, 1)  # get face frame width
    height = np.size(img, 0)  # get face frame height
    left_eye = None
    right_eye = None
    for (x, y, w, h) in eyes:
        if y > height / 2:
            pass
        eyecenter = x + w / 2  # get the eye center
        if eyecenter < width * 0.5:
            left_eye = img[y:y + h, x:x + w]
            cv2.rectangle(img,(x,y),(x+w,y+h),(0,225,255),2)
        else:
            right_eye = img[y:y + h, x:x + w]
            cv2.rectangle(img,(x,y),(x+w,y+h),(0,225,255),2)

    return left_eye, right_eye


face_frame = detect_faces(img, face_cascade)
if face_frame is not None:
    eyes = detect_eyes(face_frame, eye_cascade)



cv2.imshow('my image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Step 2: key Generation

```
distVector = []
distVector.append(np.sqrt( (center_face[0]-rightEyes_upper[0])**2 + (center_face[1]-
rightEyes_upper[1])**2 ))
distVector.append(np.sqrt( (center_face[0]-rightEyes_bottom[0])**2 + (center_face[1]-
rightEyes_bottom[1])**2 ))
distVector.append(np.sqrt( (center_face[0]-leftEyes_upper[0])**2 + (center_face[1]-
leftEyes_upper[1])**2 ))
```

```
distVector.append(np.sqrt( (center_face[0]-leftEyes_bottom[0])**2 + (center_face[1]-
leftEyes_bottom[1])**2 ))
distVector.append(np.sqrt( (center_face[0]-center_rightEyes[0])**2 + (center_face[1]-
center_rightEyes[1])**2 ))
distVector.append(np.sqrt( (center_face[0]-center_leftEyes[0])**2 + (center_face[1]-
center_leftEyes[1])**2 ))

distVector

'''
Variance and standard deviation is calculated for pairwise Euclidean distance
'''
import statistics

variance = statistics.variance(distVector)
std_dev = statistics.stdev(distVector)
print("Variance of sample set is % s"  %(variance))
print("Standard Deviation of sample is % s " %(std_dev))

'''
Unique 3 digit key generation
Key= (Most Significant 2 Digits of p1) * 10 + (Least Significant 1 Digits p2)
Where
P1 = variance of Hand points pairs wise distance
P2 = standard deviation of Hand points pairs wise distance

'''
MSB = int(str(variance)[:2])
LSB = int(str(std_dev)[-1:])
#print(MSB,LSB)
key = (MSB)* 10 + (LSB)
print("Key Generated:- ",key)
```

Step 3: Encrypt Message using Transposition Cipher

```
Number = key
Sum = 0
while(Number > 0):
    r = Number % 10
    Sum = Sum + r
    Number = Number//10
print(Sum)
face_key = Sum


#Transposition cipher

def encryptMessage(key, message):
    ciphertext = [''] * key

    for col in range(key):
        position = col
        while position < len(message):
            ciphertext[col] += message[position]
            position += key
    return ''.join(ciphertext) #Cipher text

myMessage = input("Enter data to be encoded(Without Space) : ")
#myMessage = "Transposition Cipher"

ciphertext = encryptMessage(face_key, myMessage)

print("Cipher Text is")
print(ciphertext + '|')
#pyperclip.copy(ciphertext)
```

```
Output:- Enter data to be encoded(Without Space) : Anu-radh@ 123
Cipher Text is
A1n2u3-radh@
```

Step 4: Image Steganography using LSB

```
def to_bin(data):
    """Convert `data` to binary format as string"""
    if isinstance(data, str):
        return ''.join([ format(ord(i), "08b") for i in data ])
    elif isinstance(data, bytes) or isinstance(data, np.ndarray):
        return [ format(i, "08b") for i in data ]
    elif isinstance(data, int) or isinstance(data, np.uint8):
        return format(data, "08b")
    else:
        raise TypeError("Type not supported.")


'''
- First carrier image or cover image has been read and converted in to array of bits
- the secret message which is in the form of characters are converted in to the “ASCII”
values
- ASCII values are converted in to array of bits
- key will gives unique pixel position of cover image to embed the data LSB of particular
image pixel of cover object

Output: Stego-image has been generated which contains secret message embedded
within cover image.

'''
#res = input("Enter data to be encoded(Without Space) : ")
res= str(ciphertext)
print("res",res)

#data = ''.join(format(ord(i), 'b') for i in res)
#print("The string after binary conversion : " + data)
data = to_bin(res)
msg_len = len(data)
print("msg length",msg_len)


i=0

with Image.open("source_img.png") as img:

    width, height = img.size
    print(width)
    print(height)
    #img.show()
    print(key)
    for x in range(key, width):
        for y in range(key, height):
            pixel = list(img.getpixel((x, y)))
            for n in range(0,3):
                if(i < len(data)):
                    pixel[n] = pixel[n] & ~1 | int(data[i])
                    i+=1

            img.putpixel((x,y), tuple(pixel))


    img.save("source_secret.png", "PNG")
```

Step 5: Data Extraction using LSB

```
'''
- Embedded secret data has extracted from stego-image
- Extracting secret data the hand key is used

'''
extracted_bin = []
with Image.open("source_secret.png") as img:
    width, height = img.size
    byte = []
    for x in range(key, width):
        for y in range(key, height):
            pixel = list(img.getpixel((x, y)))
            for n in range(0,3):
                extracted_bin.append(pixel[n]&1)

data = "".join([str(x) for x in extracted_bin])
msg =''

for i in range(msg_len):
    msg+=str(extracted_bin[i])
#print(msg)

decoded_msg = ''.join(chr(int(msg[i*8:i*8+8],2)) for i in range(len(msg)//8))
print("decoded_msg:= ",decoded_msg)
```

```
Output:- decoded_msg:=  A1n2u3-radh@
```

Step 6: decrypt msg using Transposition Cipher

```
import math
def decryptMessage(key, message):

    numOfColumns = math.ceil(len(message) / key)
    numOfRows = key
    numOfShadedBoxes = (numOfColumns * numOfRows) - len(message)
    plaintext = [""]  * numOfColumns
    col = 0
    row = 0

    for symbol in message:
        plaintext[col] += symbol
        col += 1
        if (col == numOfColumns) or (col == numOfColumns - 1 and row >= numOfRows - 
numOfShadedBoxes):
                col = 0
                row += 1
    return ''.join(plaintext)

plaintext = decryptMessage(face_key, decoded_msg)
print("The plain text is")
print(plaintext)
```

```
Output:- The plain text is : Anu-radh@ 123
```

Chapter 5: Results

Hand Geometry

Table 1 shows that the results of proposed system. Here the distortion of original cover image and stego image are calculated using PSNR. As the message length increases the system's response time increases and PSNR value decreases.

Table 1

Msg Length	**PSNR**
50	99.938
100	97.039
200	94.149
400	91.155
500	90.75
1000	87.80
2000	84.80
4000	81.78
8000	78.80

Face Geometry

Table 2 shows that the results of proposed system. Here the distortion of original cover image and stego image are calculated using PSNR. As the message length increases the system's response time increases and PSNR value decreases.

Msg Length	**PSNR**
50	99.057
100	96.047
200	93.037
400	90.062
500	89.150
1000	86.051
2000	83.075
4000	80.065
8000	77.064

Future Scope

1. The system uses only the LSB algorithm to hide the data in image, a number of algorithms exist which can also be implemented. The user can be given a list of algorithms to choose among according to his/her application.
2. A graphical user interface can be added, so that non-programmers can also use it as a tool.
3. It can be implemented as a downloadable python package, so that python programmers can import and use it as a tool in their applications.
4. The system hides only text in the image, we can also add functionality to hide image using this technique.
5. The system uses hand and face image for key generation, we can also use other biometrics such as fingerprint, Skin tone etc.

References

1. Priya Yankanchi, Shanmukhappa A. Angadi “BIOMETRIC STEGANOGRAPHY: A NEW APPROACH USING HAND GEOMETRY”, International Journal of Latest Trends in Engineering and Research (IJLTER) ,ISSN-Online:2455-1457

2. https://www.pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/

3. https://matplotlib.org/

4. https://pythonprogramming.net/loading-images-python-opencv-tutorial/

5. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html