Cover Page

EEP3010: Communication Lab Project Report

on

Secure Message Transmission using encryption and dream catcher transmission kit.

By

Ghelani Shubham- B20EE019

H Sriram- B20EE020

Haardik Ravat- B20EE021

Abstract

Steganography is a technique of hiding secret information within an image in a way that the presence of the information is concealed. The process of steganography involves converting an image into binary digits, embedding a secret message within the binary data, and transferring the data using a digital modulation scheme such as BPSK. The Dream Catcher training kit is a tool that is used to train and develop digital communication systems, including BPSK modulation. The kit helps individuals to learn the skills and knowledge required to use digital modulation techniques to transfer data over communication channels. Using steganography, individuals can hide messages within images for covert communication while maintaining high levels of security and confidentiality.

Equipments

For the experiment the following apparatus are required:

- N9010A Signal Analyzer/89601 Vector Signal Analysis Software with Oscilloscope (Digitizer)
- AFG3021B Function Generator (2 set)
- Dream Catcher transmitter and receiver trainer kit
- ME1100 dream-catcher training kit
- A PC with MATLAB/LABVIEW
- Coaxial cables and USB cables
- Antenna
- Local oscillator signal generator
- SMA(m)-to-SMA(m) coaxial cable

Functions of Equipments used:
The 89601 Vector Signal Analysis Software is a software package that runs on a computer and provides advanced analysis and visualization tools for RF signals. The dream catcher kit is used for transmission and receiving the receiving the RF signal. The dream catcher kit contains both active and passive components such as upconverter, Power Amplifier and RF band pass filter.

Theory

Secure Message Transmission using encryption and dream catcher transmission kit is a project that involves two key concepts: steganography and dream catcher transmission technology. In this project, a secret message is first encrypted and then embedded in an image using steganography. The resulting steganographic image is then transmitted through the dream catcher transmission kit, which is a device that converts the digital data into an analog signal that can be transmitted over a wireless channel using the electromagnetic waves. At the receiver end, the analog signal is converted back into digital data and the embedded message is extracted using steganography and decrypted using the same encryption algorithm.

Steganography is the technique of hiding secret information within an ordinary, non-secret file or message to avoid detection by unintended recipients. In the context of this project, steganography involves embedding the encrypted message within an image file. The goal of steganography is to make the presence of the embedded message as inconspicuous as possible, such that the image file appears to be normal and unmodified to anyone who may intercept or view it. The method used to embed the message depends on the type of file format used, but it typically involves altering the least significant bits of the image's pixels or color components. In this project we are moidying the least significant bit of the image to hide the message.

To use the dream catcher transmission kit for secure message transmission, the sender first encrypts the message and then embeds it in an image file using steganography. The resulting steganographic image is then transmitted via the dream catcher kit. At the receiver end, the analog signal is received and converted back into digital data. The embedded message is then extracted from the steganographic image using steganography and decrypted using the same encryption algorithm and key used by the sender.

In summary, Secure Message Transmission using encryption and dream catcher transmission kit involves embedding an encrypted message within an image file using steganography, transmitting the resulting steganographic image using the dream catcher transmission kit, and extracting and decrypting the message at the receiver end using the same encryption and steganography techniques. The resulting system provides a secure and covert means of transmitting sensitive information over a wireless channel.

Experimental Procedure

The experiment is divided into 2 component- **hardware** and **software** component. The software component involves embedding the text into image, converting the image to binary image and applying BPSK modulation. In the receiver side the opposite is performed- BPSK demodulation, converting binary string to image and then decoding text from image.
In the hardware component first the wave goes through upconverter, bandpass filter and is amplified before being sent through the channel. After the wave has been received it goes through band pass filter, low noise amplifier before being downconverted.

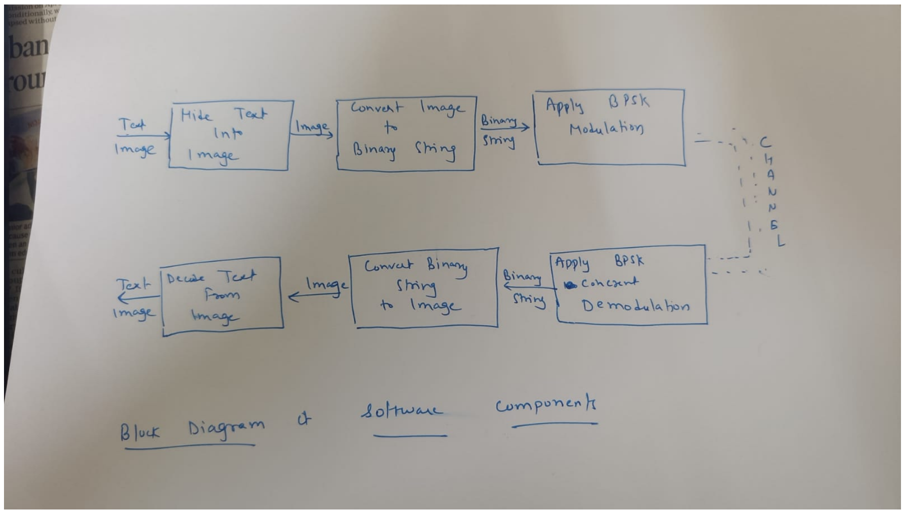

Text
Image
Hide Text Into Image
Image
Convert Image to Binary String
Binary String
Apply BPSK Modulation
CHANNEL
Apply BPSK Coherent Demodulation
Binary String
Convert Binary String to Image
Image
Decode Text From Image
Text
Image
Block Diagram of Software Components

Figure 1: Block Diagram of software component

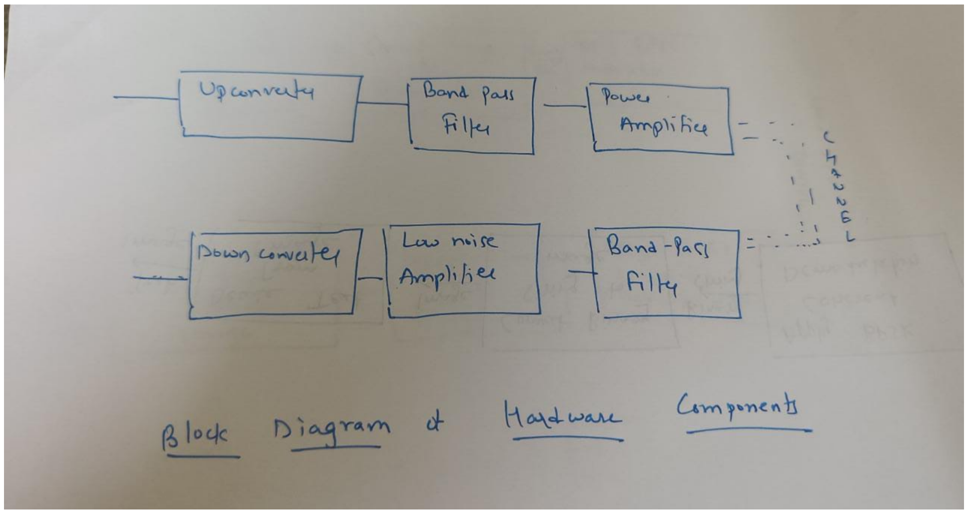


Figure 2: Block digram of hardware component

Analysis of steganography code

Encoding side

```python
# Convert encoding data into 8-bit binary
# form using ASCII value of characters
def genData(data):

        # list of binary codes
        # of given data
        newd = []

        for i in data:
            newd.append(format(ord(i), '08b'))
        return newd

# Pixels are modified according to the
# 8-bit binary data and finally returned
def modPix(pix, data):

    datalist = genData(data)
    lendata = len(datalist)
    imdata = iter(pix)

    for i in range(lendata):

        # Extracting 3 pixels at a time
        pix = [value for value in imdata.__next__()[:3] +
                                imdata.__next__()[:3] +
                                imdata.__next__()[:3]]

        # Pixel value should be made
        # odd for 1 and even for 0
        for j in range(0, 8):
            if (datalist[i][j] == '0' and pix[j]% 2 != 0):
                pix[j] -= 1

            elif (datalist[i][j] == '1' and pix[j] % 2 == 0):
                if(pix[j] != 0):
                    pix[j] -= 1
                else:
                    pix[j] += 1
                # pix[j] -= 1

        # Eighth pixel of every set tells
        # whether to stop ot read further.
        # 0 means keep reading; 1 means thec
        # message is over.
        if (i == lendata - 1):
            if (pix[-1] % 2 == 0):
                if(pix[-1] != 0):
                    pix[-1] -= 1
                else:
                    pix[-1] += 1

        else:
            if (pix[-1] % 2 != 0):
                pix[-1] -= 1

        pix = tuple(pix)
        yield pix[0:3]
        yield pix[3:6]
        yield pix[6:9]

def encode_enc(newimg, data):
    w = newimg.size[0]
    (x, y) = (0, 0)

    for pixel in modPix(newimg.getdata(), data):

        # Putting modified pixels in the new image
        newimg.putpixel((x, y), pixel)
        if (x == w - 1):
            x = 0
            y += 1
        else:
            x += 1

# Encode data into image
def encode():
    img = input("Enter image name(with extension) : ")
    image = Image.open(img, 'r')

    data = input("Enter data to be encoded : ")
    if (len(data) == 0):
        raise ValueError('Data is empty')

    newimg = image.copy()
    encode_enc(newimg, data)

    new_img_name = input("Enter the name of new image(with
    extension) : ")
    newimg.save(new_img_name, str(new_img_name.split(".")[1].upper
    ()))
```

Listing 1: Encoding code

• This code is an implementation of a basic Steganography algorithm in Python using the PIL (Python Imaging Library) module. The code contains three main functions: genData(), modPix(), and encode(). The genData() function takes the message to be hidden as input and returns a list of 8-bit binary codes of the ASCII values of the message characters. This is done by iterating over each character in the message, converting its ASCII value to an 8-bit binary string using the format() function, and adding it to the list.

• The modPix() function takes the pixel data of the image and the binary codes of the message characters as input. It modifies the least significant bits of the pixel values to store the binary codes of the message characters. It does this by iterating over the binary codes of the message characters and extracting three pixels at a time. It then modifies the least significant bits of these pixels according to the binary code of the corresponding message character. Finally, it yields the modified pixel values.

• The encode() function takes the input image file name and the message to be hidden as input. It first opens the input image file using the Image.open() function of the PIL module. It then creates a copy of the input image using the copy() function of the Image class. The modPix() function is then called with the pixel data of the copy image and the binary codes of the message characters as input to modify the pixels to store the binary codes of the message characters. Finally, the modified image is saved with the desired name using the save() function of the Image class.

Converting Image to Binary string

```
#Function to Convert an image to binary string
def image_to_binary_string(image_file):
    # Open the image file
    image = Image.open(image_file)

    # Convert the image to a binary string by converting the pixels
     into ascii format
    binary_string = ''.join(format(byte, '08b') for pixel in image.
    getdata() for byte in pixel)
    return binary_string

# Example usage: convert an image to a binary string
image_file = "new_data.png"
binary_string_op = image_to_binary_string(image_file)
print(len(binary_string_op))

```

Listing 2: Image to Binary string

In the following code the individual pixels of the modified image are broken down into its RGB values and the RGB values are converted to binary format by using the ASCII format. These ASCII values are joined together to get the binary string.

Analysis of transmitter code

```

N=2^6; % Number of bits = 64
%% text to binary
fid = fopen('text_to_string.txt');
    x=fread(fid,'*char');
    disp(length(x))
    binary = dec2bin(x,1);   % The number of bits is taken as 1
    since the text is already in binary format
    disp(binary)
    binary_t=transpose(x);
    txt_to_bin=binary_t(:)-'0';


%% adding syncronization bits
h_pn=commsrc.pn('GenPoly',[6 5 0],'InitialStates',[0 0 0 0 0 1],'
    NumBitsOut',N);
syncronization_bits=generate(h_pn);
messageLength = dec2bin(length(txt_to_bin), 16);  %defining message
     length
messageLengthTransposed = transpose(messageLength);
text_to_bin_messageLength = messageLengthTransposed(:) - '0';


%% [sync bits, message length, data bits]
tx_text = [syncronization_bits; text_to_bin_messageLength;
    txt_to_bin];


%% Bit Sequence to Signal (Upsampling by 8)
bits_I = 2*(tx_text - 0.5);
sig_ipt_I = upsample(bits_I, 8);
FIR_coeff75 = Hps75.Numerator;
sig_RRC75_I = conv(FIR_coeff75,sig_ipt_I);
sig_RRC75_I = sig_RRC75_I/max(sig_RRC75_I);
disp(length(sig_RRC75_I))
csvwrite('Initial_data.csv',tx_text);
csvwrite('Transmit_data.csv',sig_RRC75_I);

```

Listing 3: trasnmitter code

In MATLAB a PRBS(Psuedo Random Binary Sequence) is generated. The PRBS signal is not a noise sequence but a code which resembles a noise sequence. The signal is generated using a Linear Feedback Shift Register. Using the command *commsrc* the PRBS signal is generated. The order of the polynomial and the initial states of the LFSR is given as parameters. The number of bits generated is 2^6. The text file is then loaded as a binary sequence. To convert in into a double format we subtract a '0' from it and

then synchronised using the PRBS sequence. The synchronization tells us about the starting bit of the message. To find the end of the message we use the length of the text file as parameter itself.
The sequence is then upsampled to 8 bits and then pulse shaping is done using a root raised cosine filter and then convolution is done followed by normalisation.The waveforms are then converted to a csv file and imported in a National Instrument Machine.

Hardware Procedure
Since this is a BPSK experiment only I data is generated. The I data is fed into a function generator. In the function generator we load the user data and then set the Burst frequency. The burst frequency is given by:

$$Burst\ Freq = \frac{Transmission\ Rate\ of\ I\ data}{Length\ of\ I\ data}$$

The transmission rate is 200KHz symbols/sec and each symbol is of 8 bits. Therefore the transmission rate is 1600 Kbits/sec. THe Length of I data is 2796 samples.

$$Burst\ Freq = \frac{160000}{12800} = 125Hz$$

.The amplitude is set as 250mVpp and the interval is set for the burst frequency. The trigger interval is given by:

$$Trigger\ Interval = \frac{1}{Burst\ Freq} = 8.00ms$$

In trigger interval 0.05ms variation is added.

$$Trigger\ Interval = 8.00 + 0.02 = 8.02ms$$

The output of the function generator is fed to a modulator.The Q port is terminated using a terminator. In the LO port a carrier wave of frequency 50MHz and -3DB is fed. The RF port is then fed to a dream catcher kit In the dream catcher the signal is then upconverted to 868Mhz using a carrier frequency of 818Mhz. The signal is then amplified using a power amplier

Upconversion is required because as the frequency increases the lenght of the antenna decreases as length of antenna is directly proportional to wavelength. Hence the wavelength for 868Mhz is much less compared to wavelength for 50Mhz. Hence the antenna size 868Mhz is much smaller compared to antenna size for 50Mhz.

$$Antenna\ Size \propto \lambda(wavelength)$$

Power Amplification is done because in free space power dissipates according to the relation

$$Power \propto \frac{1}{radius^2}$$

Hence to transmit over a distance we need to make sure that our signal has sufficient power. Hence Power amplification is done. In the receiver side the reverse procedure is done. The signal is first passe through a low noise amplifier. Then is it donwconverted to 50Mhz using a carrier wave of 818 Mhz. It is then passed through a low bandpass filter. The signal is then connected to the NI instrument to view its wave.

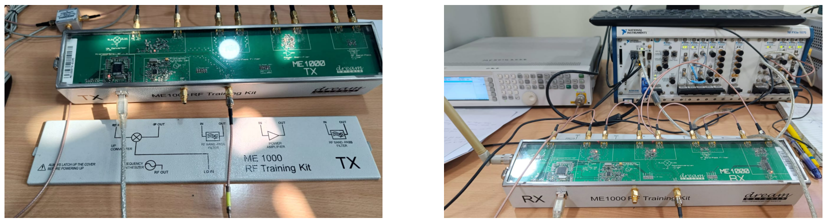


Figure 3: Dream Catcher Transmitter and receiver kit

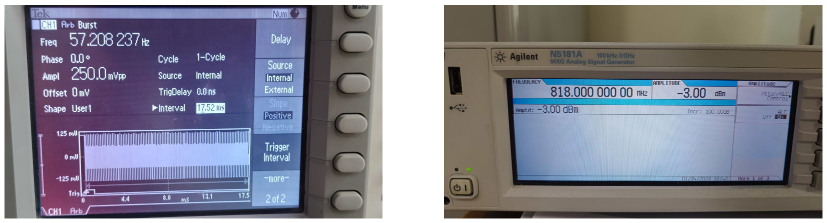


Figure 4: Function generators

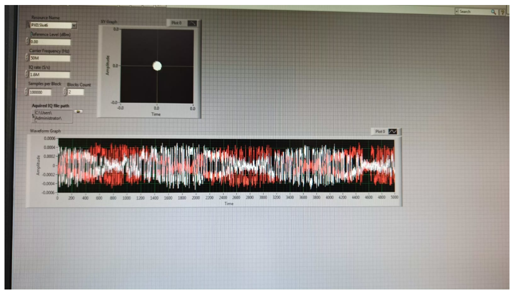


Figure 5: IQ data at receiver end

Analysis of Receiver code

```
N = 2^6;
h_pn = commsrc.pn('GenPoly', [6 5 0], 'InitialStates', [0 0 0 0 0
    1], 'NumBitsOut', N);
syncronization_bits = generate(h_pn);

%% Reading data file and using root raised cosine filter with roll
    off value as 0.75

samplesPerSymbol = 8; % Samples per symbol
RX_ss_tmp = csvread('received.csv',0,0); % Acquired data at the NI
    using LabView (RFSA acquired)
RX_ss = complex(RX_ss_tmp(:,1), RX_ss_tmp(:,2)); % Received data
    having two column (I and Q) converting into complex

%Filtering received signal using root raised cosing filter with
    rollof factor as 0.75
FIR_coeff75 = Hps75.Numerator;
XX_signal = conv(FIR_coeff75, RX_ss);
RX_signal = XX_signal/max([real(XX_signal);imag(XX_signal)]);


%% EYE Diagram
eye_len = samplesPerSymbol;
eye_frame_len=floor(length(RX_signal)/eye_len);
I_eye=zeros(eye_len,eye_frame_len);
Q_eye=zeros(eye_len,eye_frame_len);

for i=1:eye_frame_len
    I_eye(:,i)= real(RX_signal((i-1)*eye_len+1:i*eye_len,1));
    Q_eye(:,i)=imag(RX_signal((i-1)*eye_len+1:i*eye_len,1));
end





%% Sampling Instant Ientification
eye_var=zeros(eye_len,1);
for i=1:eye_frame_len
    for j=1:eye_len
        eye_var(j,1)=eye_var(j,1)+I_eye(j,i)^2;
    end
end
eye_var=eye_var/eye_frame_len;
[~,eye_offset]=max(eye_var(1:samplesPerSymbol));

%% Downsampling of the filtered signal by 8 at the best sampling
    instant
Symbols=zeros(floor(length(RX_signal)/samplesPerSymbol),1);
for i=1:length(Symbols)-1
    Symbols(i,1)=RX_signal((i-1)*samplesPerSymbol+eye_offset,1);
end

```

```
%% Implementation of phase and frequency correction(PLL/Costas loop
    ) for demodulation
K1_PLL =0.0313;
K2_PLL =2.49e-4;
unwrap_phi_array = zeros(size(Symbols));
phi_array = zeros(size(Symbols));
freq_array = zeros(size(Symbols));
filt_phi_array = zeros(size(Symbols));
filt_freq_array = zeros(size(Symbols));
phi_array(1:2)=atan(imag(Symbols(1:2))./real(Symbols(1:2)));
unwrap_phi_array(1:2)=phi_array(1:2);
for i=3:length(Symbols)
    phi=atan(imag(Symbols(i))/real(Symbols(i)));
    phi1(i)=phi;
    old_phi=phi_array(i-1);
    if (phi-old_phi < -pi/2)
        freq = pi+phi-old_phi;
    elseif (phi-old_phi > pi/2)
        freq = -pi+phi-old_phi;
    else
        freq = phi-old_phi;
    end
    freq_array(i)=freq;
    unwrap_phi_array(i)=unwrap_phi_array(i-1)+freq;
    phi_array(i)=phi;
    filt_phi_array(i) = (2-K1_PLL-K2_PLL)*filt_phi_array(i-1) ...
        -(1-K1_PLL)*filt_phi_array(i-2) ...
        +(K1_PLL+K2_PLL)*unwrap_phi_array(i-1) ...
        -(K1_PLL)*unwrap_phi_array(i-2);
    Symbols(i) = Symbols(i)*complex(cos(filt_phi_array(i)),-sin(
    filt_phi_array(i)));


end



%~~~~~~~~~Finding the starting bit and detect the bits (Works in
    good SNR)~~~~~~~~~~~~~~~~~~~~~~~%
bits_rec_bipolar=sign(real(Symbols));
corval=zeros(length(bits_rec_bipolar),1);

%% Correlation value to get starting index
%~~~~~~~~~~~~~~~ Using syncronization bit finding stating start of
    the data packet~~~~~~~~~~~~~~~%
for i=1:(length(bits_rec_bipolar)-64) %subtracting 64 to prevent
    overflow when iterating through bits_rec_bipolar
    corval(i,1)=sum(syncronization_bits(1:64).*bits_rec_bipolar(i:i
    +63));
end

[peak,start_bit_ind]=max(abs(corval)); %searching maximum
    correlation value and its index number
start_bit_ind
peak

ss=sign(corval(start_bit_ind));       % required to correct the
```

```
    polarity of PLL output
bits_rec=(ss*bits_rec_bipolar+1)/2;   %converting 1--->1 and
    -1---->0

len1 = bits_rec(start_bit_ind + 64 : start_bit_ind + 80);
len = string(len1);

sLen = "";
for i=1:length(len)-1
    sLen = sLen + len(i);
end
len = bin2dec(sLen);


detected_bits = bits_rec(start_bit_ind:start_bit_ind + 79 + len); %
     chopping out high correlted length

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Saving the binary data to a
     text ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
bin_to_text = detected_bits;
bin_to_text = bin_to_text(81:end);
bin_to_text;
bin_str=sprintf('%d',bin_to_text)
 %Open the file for writing
fid = fopen('text_to_string.txt', 'w');

%Print the string variable to the file
fprintf(fid, '%s', regexprep(bin_str, '\s', ''));


```

Listing 4: Receiver code

In the receiver side same procedure is followed as transmitted side. The data is read using csv read file and converted to complex form.A rectangular filter is generated. The data is then filtered and convolved and then normalised. The eye diagram for the I data is gernerated in MATLAB. THe signal is then downsampled by 8. Next phase correction of symbol variable is done using costas loop. The phase of I data is calculated using atan matlab command. The phase correction is done after that.

Converting Binary String to Image

```python
#Function to convert binary string to image
from PIL import Image

def binary_string_to_image(binary_string, image_size):
    # Create a new image with the specified size
    image = Image.new('RGB', image_size)

    # Set the pixel values in the image from the binary string
    pixel_data = []
    for i in range(0, len(binary_string), 24):
    #Set the RGB values of each pixel
        red = int(binary_string[i:i+8], 2)
        green = int(binary_string[i+8:i+16], 2)
        blue = int(binary_string[i+16:i+24], 2)
        pixel_data.append((red, green, blue))
    image.putdata(pixel_data)

    return image

# Example usage: convert a binary string to an image
image_size = (9, 7)
image = binary_string_to_image(binary_string_op, image_size)
image.save("decode_image.png")

```

Listing 5: Image to Binary string

In the following code the binary string is converted to RGB value by combing 8 bits together and forming its ASCII value. Once the RGB values are formed they are combined to form a pixel. Once the pixels are decoded , by using the size of the image mentioned the image is reconstructed.

Decoding text from Image

```python
# Decode the data in the image
def decode():
    img = input("Enter image name(with extension) : ")
    image = Image.open(img, 'r')

    data = ''
    imgdata = iter(image.getdata())

    while (True):
        pixels = [value for value in imgdata.__next__()[:3] +
                                imgdata.__next__()[:3] +
                                imgdata.__next__()[:3]]

        # string of binary data
        binstr = ''

        for i in pixels[:8]:
            if (i % 2 == 0):
                binstr += '0'
            else:
                binstr += '1'

        data += chr(int(binstr, 2))
        if (pixels[-1] % 2 != 0):
            return data

```

Listing 6: Image to Binary string

The image data is read pixel by pixel and for each bit RGB value if the value is even it means that the value of the bit was 0 and if it was odd then it means that the value of the bit was 1. For each character 8 bits are taken as its ascii value and the character is decoded. After every 8 bit if the value of teh 9th bit is odd it means that more characters are left to be decoded, if it is even then the algorithm stops at that instant.

Experimental Results

Initially we are taking the image of **9x7** size. The length of the message to be encoded is taken as **arm**.

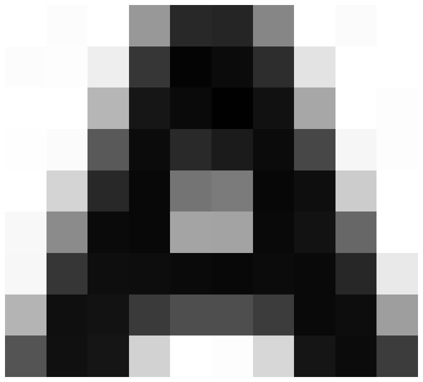
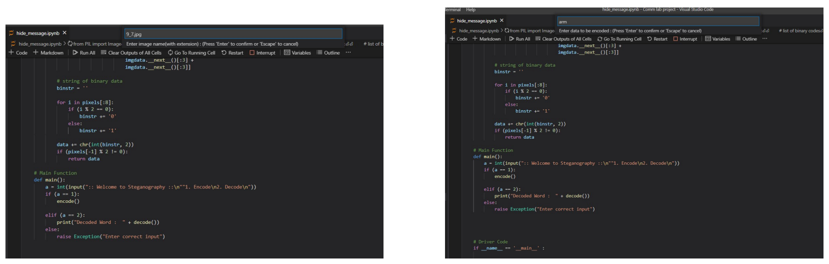


Figure 6: Original image



```
9_7.jpg
Enter image name(with extension) : (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
                            imgdata.__next__()[:3] +
                            imgdata.__next__()[:3]]

        # string of binary data
        binstr = ''

        for i in pixels[:8]:
            if (i % 2 == 0):
                binstr += '0'
            else:
                binstr += '1'

        data += chr(int(binstr, 2))
        if (pixels[-1] % 2 != 0):
            return data

# Main Function
def main():
    a = int(input(":: Welcome to Steganography ::\n""1. Encode\n2. Decode\n"))
    if (a == 1):
        encode()

    elif (a == 2):
        print("Decoded Word :  " + decode())
    else:
        raise Exception("Enter correct input")
```

Terminal
Help
hide_message.ipynb - Comm lab project - Visual Studio Code
hide_message.ipynb
hide_message.ipynb > from PIL import Image
+ Code
+ Markdown
Run All
Clear Outputs of All Cells
Go To Running Cell
Restart
Interrupt
Variables
Outline
# list of binary codes

```
arm
Enter data to be encoded : (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
                            imgdata.__next__()[:3] +
                            imgdata.__next__()[:3]]

        # string of binary data
        binstr = ''

        for i in pixels[:8]:
            if (i % 2 == 0):
                binstr += '0'
            else:
                binstr += '1'

        data += chr(int(binstr, 2))
        if (pixels[-1] % 2 != 0):
            return data

# Main Function
def main():
    a = int(input(":: Welcome to Steganography ::\n""1. Encode\n2. Decode\n"))
    if (a == 1):
        encode()

    elif (a == 2):
        print("Decoded Word :  " + decode())
    else:
        raise Exception("Enter correct input")



# Driver Code
if __name__ == '__main__' :
```

Figure 7: Image encoding part

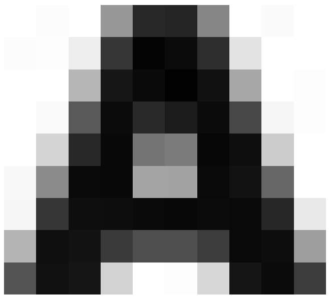
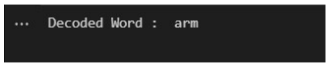


Figure 8: Encoded Image

After sending the data through the dream catcher transmitter kit and decoding it the following image and word is obtained.



```
...  Decoded Word :  arm
```

Figure 9: Decoded word

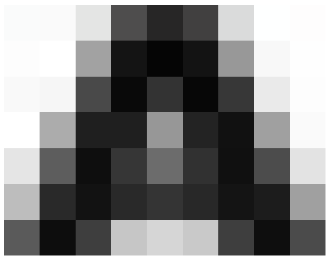
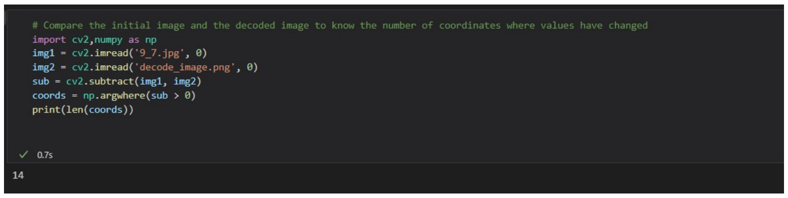


Figure 10: Decoded Image

We then compare the original image and the decoded image to find out how many pixels have changed from the original image in the decoded image.



```
# Compare the initial image and the decoded image to know the number of coordinates where values have changed
import cv2,numpy as np
img1 = cv2.imread('9_7.jpg', 0)
img2 = cv2.imread('decode_image.png', 0)
sub = cv2.subtract(img1, img2)
coords = np.argwhere(sub > 0)
print(len(coords))

✓ 0.7s
14
```

Figure 11: Analysis of decoded image w.r.t initial image

The word that was sent is **successfully decoded**. We found out that 14 pixels have changed in the decoded image compared to the original image.

Discussion

BPSK (Binary Phase Shift Keying) is a digital modulation technique that transmits information by changing the phase of a carrier signal. In BPSK, two phase states, 0 and 180 degrees, are used to represent binary digits, typically 0 and 1. In a BPSK system, the binary data to be transmitted is used to modulate the phase of a sinusoidal carrier signal. When a 0 is transmitted, the carrier signal is unchanged, while a 1 is transmitted by shifting the carrier signal by 180 degrees.

Performing steganography on an image involves hiding a secret message within the image's data in such a way that it remains undetectable to human observers. This is accomplished by converting the image into a binary representation, embedding the secret message within the binary data, and then transferring the data using a digital modulation scheme such as BPSK. At the receiver end, the binary data is demodulated to recover the original image data, and the hidden message is extracted using a steganography algorithm. The Dream Catcher training kit is a tool used in the training and development of digital communication systems, including BPSK modulation. By using steganography to hide messages within images, individuals can communicate covertly while maintaining a high degree of security and confidentiality.

Limitations

• Firstly, the project requires the use of small images due to the limited capacity of the function generators which can only handle around 50,000 bits of input for a single packet

• Secondly, the project does not use line coding or other data bits to reduce the data length. This could result in a larger amount of data being transferred, leading to longer transmission times and potential errors in data transfer.

• Thirdly, due to the small size of the image, the message that can be hidden within the image is also limited. This limitation could prevent the transmission of longer messages, which may not be suitable for some applications.

• The size of the original image must be known at the time of decoding to correctly reconstruct the original

• Finally, there is a risk of image distortion after sending due to the small size of the image and the need to write data over it. This could result in a loss of image quality and potential errors in the transmission of the hidden message. Overall, these limitations highlight the need for careful consideration of the application and the appropriate use of steganography techniques to ensure that the desired level of security and confidentiality is achieved.

Conclusion

In conclusion, our project demonstrates an innovative approach to secure communication. The combination of steganography, encryption, and dream catcher transmission technology provides a robust and covert mechanism for transmitting sensitive information over a wireless channel. The secret message is encoded into the image, thus securing it while transmitting it through Dreamer Catcher Kit, and decoding it on the receiver end to reconstruct the original image and decode the secret message from it.

Contribution

- Ghelani Shubham Bhaveshbhai: Writing code for embedding secret message into image and then encoding the image to a binary string. Contributed in report.
- H Sriram: Writing code for decoding the image from a binary string and then decrypting the secret text from the image. Contributed in report.
- Haardik Ravat: The connection for transmission of the bits using the dream catcher kit and other equipments. Testing the components and determining the limitations and the upper cap of bits transmission was done.