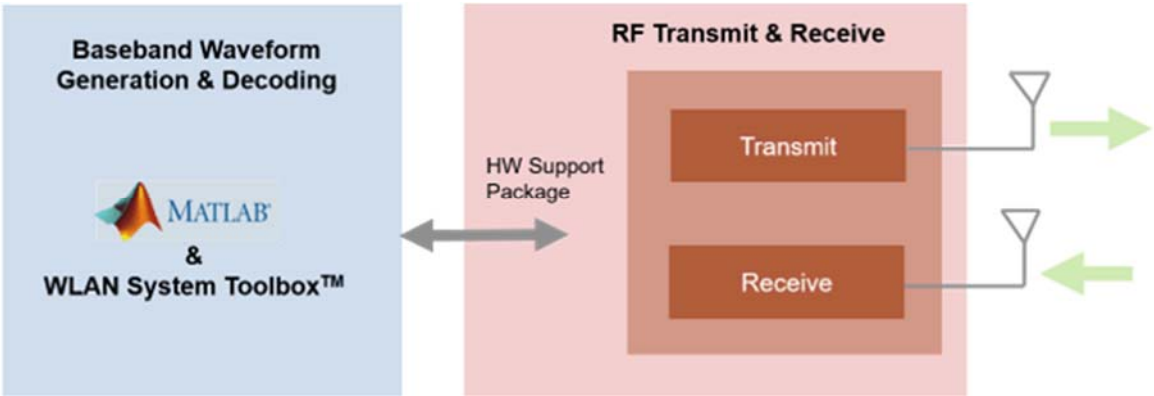
AIM: - Entropy based Image Partitioning.

INTRODUCTION: -

Image compression is one of the most important steps in image transmission and storage. Most of the state-of-art image compression techniques are spatial based. In this code, a histogram-based image compression technique is implemented based on multi-level image thresholding. The gray scale of the image is divided into crisp group of probabilistic partition. Shannon's Entropy is used to measure the randomness of the crisp grouping. The entropy function is maximized using a popular metaheuristic named Differential Evolution to reduce the computational time and standard deviation of optimized objective value.

You can use WLAN Toolbox to generate standard-compliant MAC frames and waveforms. These baseband waveforms can be upconverted for RF transmission using SDR hardware such as E3xx Radio. The Repeated Waveform Transmitter functionality with the USRP Embedded Series radio hardware allows a waveform to be transmitted over the air and is received using the same SDR hardware. The received waveform is captured and down sampled to baseband using a USRP Embedded Series Radio and is decoded to recover the transmitted information as shown in the following figure.



REQUIREMENTS: -

1. MATLAB R2019a
2. Image processing toolboxes API
3. WLAN 802.11x/ac API

MATHEMATICAL BACKGROUND

Entropy Coding / Decoding Stage: The third stage of entropy coding, determines the number of bits required to represent a particular image at a given quantization level. The process of entropy coding and decoding is lossless. It maps the quantized transform coefficients into a bit stream using variable length codes. This stage exploits the coding redundancy.[2], [5]

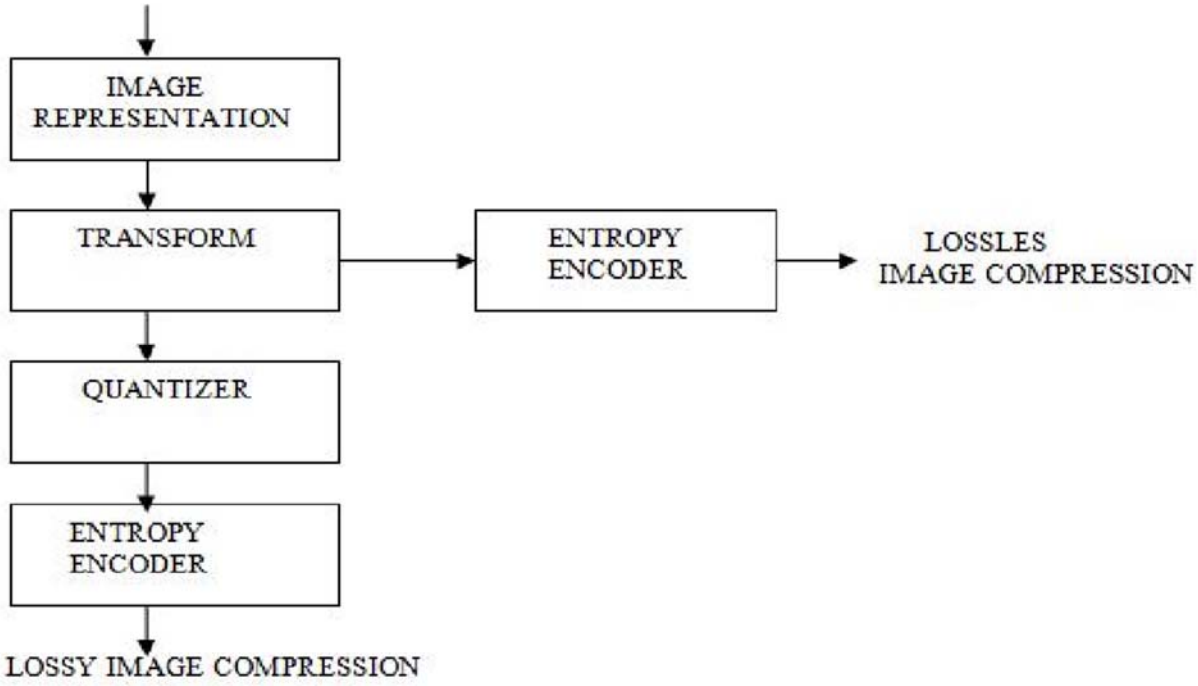


Figure 1

Now we discuss about different error-free compression approach that does not require decomposition of an image into a collection of bits planes. The approach, commonly referred to a lossless predictive coding, is based on eliminating the inter-pixel redundancies of closely spaced pixels by extracting and coding only the new information
in each pixel.[7]

2.1Huffman Coding

A commonly used method for data compression is Huffman coding. One of the most popular techniques for removing coding redundancy is due to Huffman. When coding the symbols of an information source individually, Huffman coding yields the smallest possible number of code symbols per source symbol.

2.1.1Huffman coding algorithm

In a optimum code, symbols that occurs more frequently (have higher probabilities of occurrence) will have shorter code words than symbols that occur less frequently.

The two symbols that occurs least frequently will have the same length.[10]

2.2 Entropy [8]

Entropy encoding which is a way of lossless compression that is done on an image after the quantization stage. It enables to represent an image in a more efficient way with smallest memory for storage or transmission.

The average information per source symbol is known as entropy. [8]

From a source with n symbols, the “Entropy of the source is the minimum theoretical of the average number of bit per symbol”

$$HP_i \log P_i = -\sum_{i=1}^{n} \quad (2.1)$$

P^i is the probability of the i-symbol.

Where

Given a compression scheme, its efficiency can be measured as:

Efficiency = H/(average codeword length).

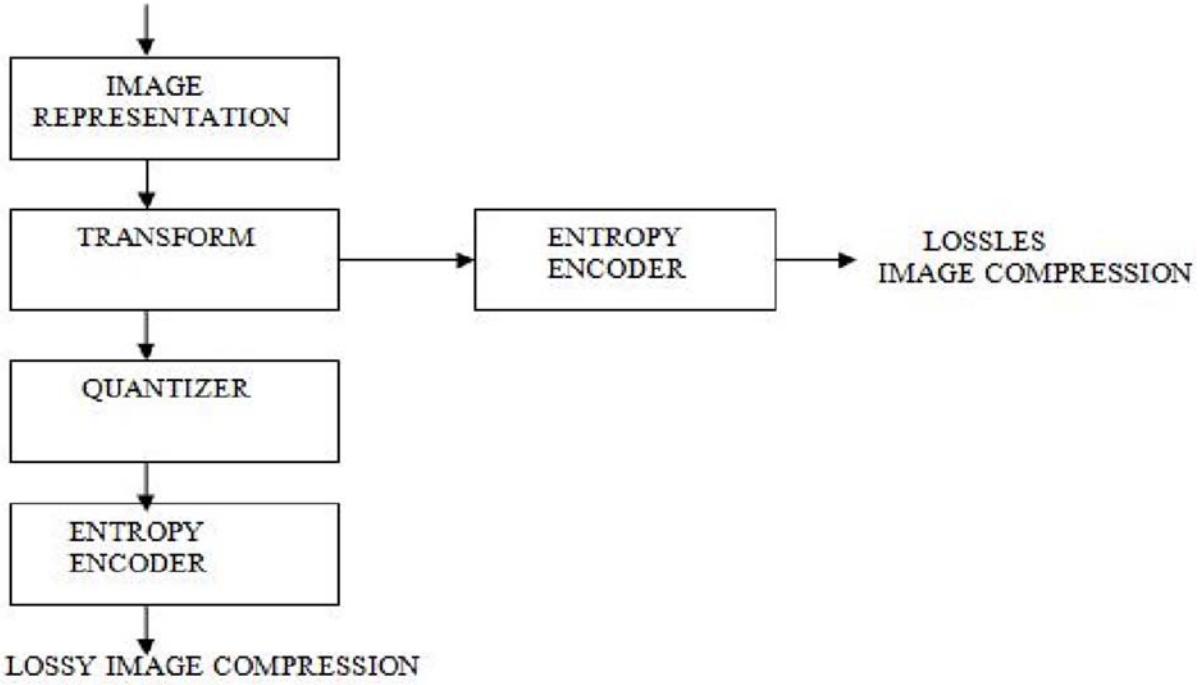
2.3 Differential entropy

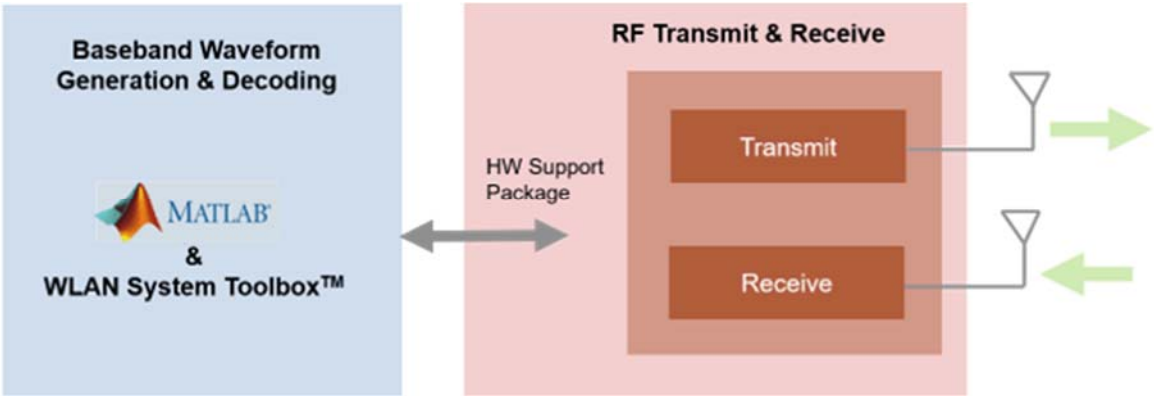
The differential entropy is given by

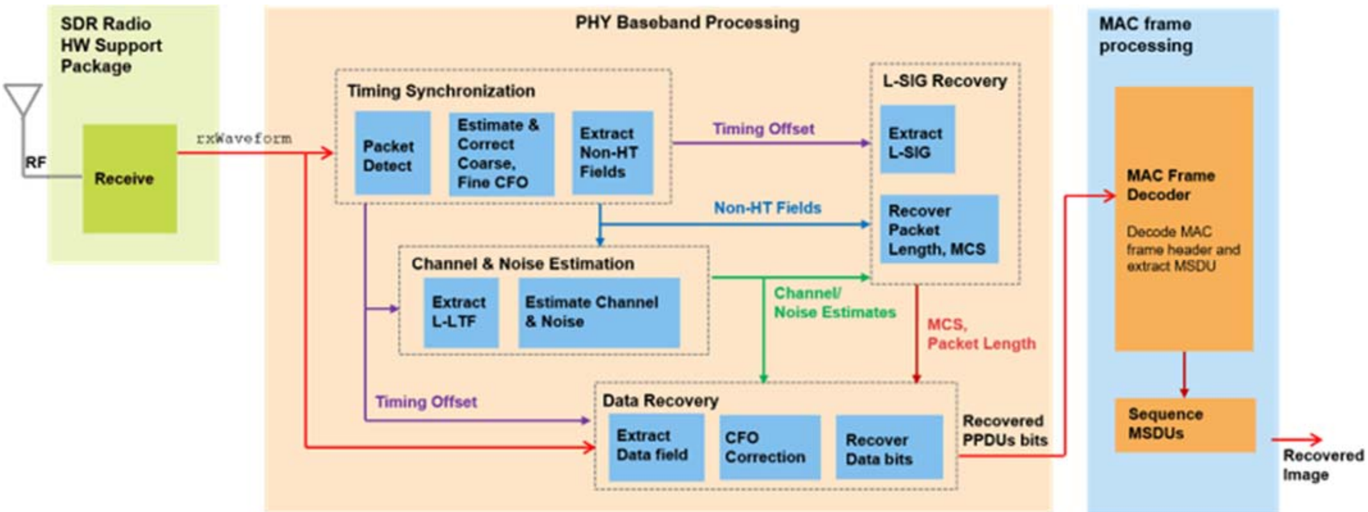
$$h(x) \leq \frac{1}{2} \log \quad 2\pi e \quad \sigma_2 \quad (2.2)$$

The differential entropy for the Gaussian distribution has the added distinction that it is larger than the differential entropy of any other continuously distributed random variable with the same variance. That is, for any random variable X, with ^{2} variance.

FLOW DIAGRAMS: -







CODE: -

```
function entropy=shannonEntropy(x,h)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
% INPUTS : x is the threshold vector
%          h is the histrogram of the image
% OUTPUT:  entropy is the Entropy of the image
after probabilistic
%          partition by the threshold vector 'h'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%


entropy=0;
x=[1 (x+1) 256];
s=size(x);

for i=1:s(2)-1
    temp = h(x(i):x(i+1));
    tsum= sum(temp);
    ent=0;
    if tsum~=0
    for j=x(i):x(i+1)
        if h(j)~=0
            a=h(j)/tsum;
```

```
            ent = ent + a*log(a);
        end
    end
    end
    entropy = entropy + ent;
end

entropy=-entropy;


function J=compressImage(I,Level)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% INPUTS : I is the input image
%          Level is the extent of compression (1 to
256)
% OUTPUT:  J is the compressed image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%


if length(size(I))>2
    I=rgb2gray(I);
end

H = imhist(I);
h = H / numel(I);
NumberOfIter = 1;
Thresholds = diffEvolution(h,Level,NumberOfIter);
Thres=[1 Thresholds+1 256];
J=0;
gray_level = zeros(1,length(Thres)-1);

for l=2:length(Thres)
    if sum(h(Thres(l-1):Thres(l)))~=0
        gray_level(l-1)=sum((h(Thres(l-
1):Thres(l)))'.*double(Thres(l-
1):Thres(l)))/sum(h(Thres(l-1):Thres(l)));
```

```
        J=J+gray_level(l-1)*(I>Thres(l-1) &
I<=Thres(l));
    end
end

J=uint8(round(J));

function T=diffEvolution(h,D,max_runs)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
% INPUTS : h is the histogram of the input image
%          D is the number of thresholds
%          max_runs is the maximum number of times
DE needs to be executed
% OUTPUT : T is the output threshold vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%



max_val=255;                                 % Upper Limit
of the search space for 8-bit images
min_val=0;                                   % Lower Limit
of the search space
range_min = min_val*ones(1,D);
range_max = max_val*ones(1,D);

%**************************************************
*******************
% Initialize the Population
%**************************************************
*******************
NP = 10*D;                                   % Population
Size of DE
maxgen = 100;                                % No. of
generations of DE
F = 0.5;                                     % Weighting
Factor
```

```
CR = 0.9;                                   % Crossover
probability
statistics_x = zeros(max_runs,D);
fitness_parent = zeros(1,NP);
fitness_child = zeros(1,NP);
x = zeros(NP,D);
w=waitbar(0,'Please Wait');

for runn = 1:max_runs

    for i = 1 : NP
        for j = 1:D
            x(i, j) =  round(range_min(j) +
((range_max(j)-range_min(j))*(rand)));
        end
        x(i,:)=sort(x(i,:));
        fitness_parent(i) =
shannonEntropy(x(i,:),h');
    end

    v = zeros(size(x));
    u = zeros(size(x));


%**************************************************
******************
    % Start of Iteration

%**************************************************
******************
    for gen = 2:maxgen

waitbar(gen/maxgen,w,sprintf('%12.0f%s',gen/maxgen*
100,'% DONE'));

%**************************************************
******************
        % Mutation
```

```
%*************************************************************
*******************
        for i = 1:NP
            r = ceil(rand(1,3)*NP);
            while r(1)==r(2) || r(2)== r(3) ||
min(r)==0 || max(r)>NP
                r = ceil(rand(1,3)*NP);
            end
            v(i,:) = x(r(1),:) + F*(x(r(2),:) -
x(r(3),:));


%*************************************************************
*******************
            % Crossover

%*************************************************************
*******************
            ra=round(rand*D);
            for j = 1:D
                if rand > CR || j==ra
                    u(i,j) = x(i,j);
                else
                    u(i,j) = v(i,j);
                end
            end
            u(i,:) = round(u(i,:));
            u(i,:)=sort(u(i,:));
        end

        for i = 1:NP
            for jj = 1:D
                u(i,jj) = max(u(i,jj),
range_min(jj));
                u(i,jj) = min(u(i,jj),
range_max(jj));
            end
```

```
            u(i,:)=sort(u(i,:));
            fitness_child(i,1) =
shannonEntropy(u(i,:),h');
        end

        for i = 1:NP
            if fitness_parent(i) < fitness_child(i)
                fitness_parent(i) =
fitness_child(i);
                x(i,:) = u(i,:);
            end
        end


        [~,globalbest_index] = max(fitness_parent);
        global_xbest = x(globalbest_index,:);

    end
    statistics_x(runn,:) = global_xbest;
end

if max_runs==1
    T =statistics_x;
else
    T = median(statistics_x);
end
close(w);



% I is the input image
% Level is the extent of compression. It should be in
between 1 and 256.
% Level=1 produces to maximum compression and Level=256
produces minimum compression
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
```

```
close all;

I = imread('image1.jpg');
Level = 116;
for i=1:size(I,3)
 cI(:,:,i) = compressImage(I(:,:,i),Level);
end
figure,subplot(1,2,1),imshow(I),title('Original
Image');
       subplot(1,2,2),imshow(cI),title('Compressed
Image BASED ON ENTROPY');
       %%

B1 = dec2bin(cI);


display('Entropy of compressed Image According to
Entropy:');
display(entropy(cI));
display('Entropy of origianl image  : ');
display(entropy(I)) ;
%%
scale = 1;                          % Image scaling factor
origSize = size(cI);               % Original input image
size
scaledSize = max(floor(scale.*origSize(1:2)),1); %
Calculate new image size
heightIx = min(round(((1:scaledSize(1))-
0.5)./scale+0.5),origSize(1));
widthIx = min(round(((1:scaledSize(2))-
0.5)./scale+0.5),origSize(2));
fData = cI(heightIx,widthIx,:); % Resize image
imsize = size(fData);                % Store new image
size
txImage = fData(:);



% Setup handle for image plot
if ~exist('imFig', 'var') || ~ishandle(imFig)
    imFig = figure;
    imFig.NumberTitle = 'off';
```

```
    imFig.Name = 'Image Plot';
    imFig.Visible = 'off';
else
    clf(imFig); % Clear figure
    imFig.Visible = 'off';
end

% Setup Spectrum viewer
spectrumScope = dsp.SpectrumAnalyzer( ...
    'SpectrumType',    'Power density', ...
    'SpectralAverages', 10, ...
    'YLimits',          [-130 -40], ...
    'Title',            'Received Baseband WLAN Signal
Spectrum', ...
    'YLabel',           'Power spectral density');

% Setup the constellation diagram viewer for equalized
WLAN symbols
constellation =
comm.ConstellationDiagram('Title','Equalized WLAN
Symbols',...

'ShowReferenceConstellation',false);

%  Initialize SDR device
deviceNameSDR = 'E3xx'; % Set SDR Device

txGain = -10;

% Plot transmit image
figure(imFig);
imFig.Visible = 'on';
subplot(211);
    imshow(fData);
    title('Transmitted Image');
subplot(212);
    title('Received image will appear here...');
    set(gca,'Visible','off');
    set(findall(gca, 'type', 'text'), 'visible', 'on');

pause(1); % Pause to plot Tx image
```

```
msduLength = 2304; % MSDU length in bytes
numMSDUs = ceil(length(txImage)/msduLength);
padZeros = msduLength-mod(length(txImage),msduLength);
txData = [txImage; zeros(padZeros,1)];
txDataBits = double(reshape(de2bi(txData, 8)', [], 1));

% Divide input data stream into fragments
bitsPerOctet = 8;
data = zeros(0, 1);

for ind=0:numMSDUs-1

    % Extract image data (in octets) for each MPDU
    frameBody =
txData(ind*msduLength+1:msduLength*(ind+1),:);

    % Create MAC frame configuration object and
configure sequence number
    cfgMAC = wlanMACFrameConfig('FrameType', 'Data',
'SequenceNumber', ind);

    % Generate MPDU
    [mpdu, lengthMPDU] = wlanMACFrame(frameBody,
cfgMAC);

    % Convert MPDU bytes to a bit stream
    psdu = reshape(de2bi(hex2dec(mpdu), 8)', [], 1);

    % Concatenate PSDUs for waveform generation
    data = [data; psdu]; %#ok<AGROW>

end

nonHTcfg = wlanNonHTConfig;         % Create packet
configuration
nonHTcfg.MCS = 6;                   % Modulation: 64QAM
Rate: 2/3
nonHTcfg.NumTransmitAntennas = 1;   % Number of
transmit antenna
chanBW = nonHTcfg.ChannelBandwidth;
nonHTcfg.PSDULength = lengthMPDU;   % Set the PSDU
length
```

```
sdrTransmitter = sdrtx(deviceNameSDR); % Transmitter
properties

% Resample the transmit waveform at 30MHz
fs = wlanSampleRate(nonHTcfg); % Transmit sample rate
in MHz
osf = 1.5;                     % OverSampling factor

sdrTransmitter.BasebandSampleRate = fs*osf;
sdrTransmitter.CenterFrequency = 2.432e9;  % Channel 5
sdrTransmitter.Gain = txGain;
sdrTransmitter.ChannelMapping = 1;         % Apply TX
channel mapping
sdrTransmitter.ShowAdvancedProperties = true;
sdrTransmitter.BypassUserLogic = true;

% Initialize the scrambler with a random integer for
each packet
scramblerInitialization = randi([1 127],numMSDUs,1);

% Generate baseband NonHT packets separated by idle
time
txWaveform = wlanWaveformGenerator(data,nonHTcfg, ...
    'NumPackets',numMSDUs,'IdleTime',20e-6, ...
    'ScramblerInitialization',scramblerInitialization);

% Resample transmit waveform
txWaveform  = resample(txWaveform,fs*osf,fs);

fprintf('\nGenerating WLAN transmit waveform:\n')

% Scale the normalized signal to avoid saturation of RF
stages
powerScaleFactor = 0.8;
txWaveform =
txWaveform.*(1/max(abs(txWaveform))*powerScaleFactor);
% Cast the transmit signal to int16, this is the native
format for the SDR
% hardware
txWaveform = int16(txWaveform*2^15);

% Transmit RF waveform
```

```
sdrTransmitter.transmitRepeat(txWaveform);

%reciever
sdrReceiver = sdrrx(deviceNameSDR);
sdrReceiver.BasebandSampleRate =
sdrTransmitter.BasebandSampleRate;
sdrReceiver.CenterFrequency =
sdrTransmitter.CenterFrequency;
sdrReceiver.OutputDataType = 'double';
sdrReceiver.ChannelMapping = 1; % Configure Rx channel
map
sdrReceiver.ShowAdvancedProperties = true;
sdrReceiver.BypassUserLogic = true;

% Configure receive samples equivalent to twice the
length of the
% transmitted signal, this is to ensure that PSDUs are
received in order.
% On reception the duplicate MAC fragments are removed.
samplesPerFrame = length(txWaveform);
requiredCaptureLength = samplesPerFrame*2;
spectrumScope.SampleRate =
sdrReceiver.BasebandSampleRate;

% Get the required field indices within a PSDU
indLSTF = wlanFieldIndices(nonHTcfg,'L-STF');
indLLTF = wlanFieldIndices(nonHTcfg,'L-LTF');
indLSIG = wlanFieldIndices(nonHTcfg,'L-SIG');
Ns = indLSIG(2)-indLSIG(1)+1; % Number of samples in an
OFDM symbol

% SDR Capture
fprintf('\nStarting a new RF capture.\n')

% Store twice the length of WLAN transmitted packet
worth of
% samples, capturedData holds requiredCaptureLength
number of baseband
% WLAN samples
capturedData = capture(sdrReceiver,
requiredCaptureLength, 'Samples');
```

```
%Reconstruct Image

if ~(isempty(fineTimingOffset)||isempty(pktOffset))&&
...
        (numMSDUs==(numel(packetSeq)-1))
    % Remove the duplicate captured MAC fragment
    rxBitMatrix = cell2mat(rxBit);
    rxData = rxBitMatrix(1:end,1:numel(packetSeq)-1);

    startSeq = find(packetSeq==0);
    rxData = circshift(rxData,[0 -(startSeq(1)-1)]);%
Order MAC fragments

    % Perform bit error rate (BER) calculation
    bitErrorRate = comm.ErrorRate;
    err = bitErrorRate(double(rxData(:)), ...

txDataBits(1:length(reshape(rxData,[],1))));
    fprintf('  \nBit Error Rate (BER):\n');
    fprintf('          Bit Error Rate (BER) =
%0.5f.\n',err(1));
    fprintf('          Number of bit errors = %d.\n',
err(2));
    fprintf('    Number of transmitted bits =
%d.\n\n',length(txDataBits));

    % Recreate image from received data
    fprintf('\nConstructing image from received
data.\n');

    decdata =
bi2de(reshape(rxData(1:length(txImage)*bitsPerOctet),
8, [])');

    receivedImage = uint8(reshape(decdata,imsize));
    % Plot received image
    if exist('imFig', 'var') && ishandle(imFig) % If Tx
figure is open
        figure(imFig); subplot(212);
    else
        figure; subplot(212);
```

```
    end
    imshow(receivedImage);
    title(sprintf('Received Image'));
end
```

OUTPUT: -

Command Window

```
Entropy of compressed Image According to Entropy:
    5.8845

Entropy of origianl image  :
    6.5207

fx >>
```

Transmitted Image

Received image will appear here...

CONCLUSION: -

1.) If we transmit uncompressed image, we see loss of Space (save longer bits of data), Energy (as we have to transmits longer bits of data)

2.) If we transmit compressed image, we see loss due to Increase in computation cost due to compression.

3. If we transmit partly compressed and partly uncompressed bits in give image we may save energy of remote device. As we partly save cost during compression and transmission.

LEARNING OUTCOMES: -

For Wireless-sensor and remote sensor where we have to save energy to increase service duration of remote device, we can compress part of the image and send partly compressed and partly uncompressed (raw bits).

REFERENCES: -

1. MATLAB with Control System, Signal Processing and Image Processing Toolboxes. wiley . https://api-wiley.ipublishcentral.com//

2. Paul, S.; Bandyopadhyay, B., "A novel approach for image compression based on multi-level image thresholding using Shannon Entropy and Differential Evolution," Students' Technology Symposium (TechSym), 2014 IEEE , vol., no., pp.56,61, Feb. 28 2014-March 2 2014
doi: 10.1109/TechSym.2014.6807914

3. https://in.mathworks.com/help/supportpkg/usrpembeddedseriesradio/ug/transmission-and-reception-of-an-image-using-wlan-system-toolbox-and-a-single-usrp-e3xx.html#d123e9836