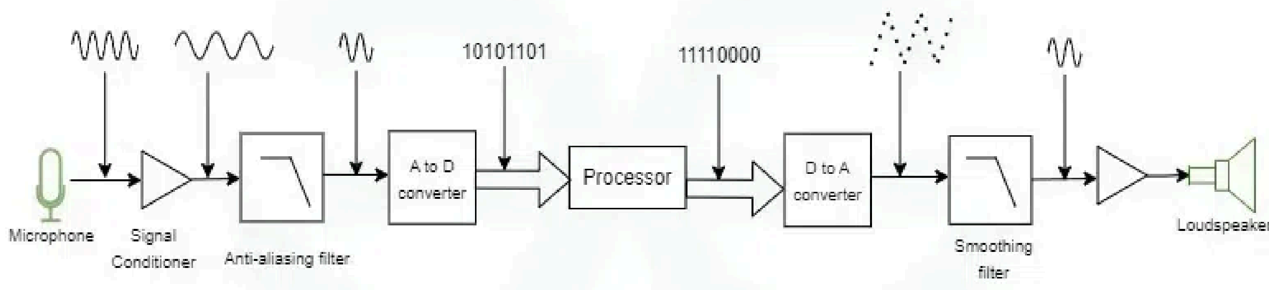
1. Removal of various types of Noise from signals acquired by sensors.

```
                         +--------------------------+
                         |   Original Signal (sin)  |
                         |    Frequency = 50 Hz     |
                         +--------------------------+
                                 |

+---------------------------+--------------------------+--------------------------+----------------------------+
           |                     |                     |                     |
           v                     v                     v                     v
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
  | White Gaussian Noise|     |   Impulse Noise    |     | Periodic Noise      |     |   Baseline Wander
|
  | Added               |     | (Salt & Pepper)    |     | (60 Hz Sinusoid)    |     | (Low-frequency drift) |
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
           |                     |                     |                     |
           v                     v                     v                     v
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
  | Noisy Signal        |     | Noisy Signal        |     | Noisy Signal        |     | Signal with Drift      |
  | (White Noise Added) |     | (Impulse Noise)     |     | (Periodic Noise)    |     |                        |
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
           |                     |                     |                     |
           v                     v                   v                     v
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
  | Low-pass Filter     |     | Median Filter       |     | Notch Filter        |     | High-pass Filter       |
  | (Cutoff: 100 Hz)    |     | (Window: 3)         |     | (59-61 Hz Bandstop) |     | (Cutoff: 0.1 Hz)       |
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
           |                     |                     |                     |
           v                   v                     v                     v
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
  | Filtered Signal     |     | Filtered Signal     |     | Filtered Signal     |     | Corrected Signal       |
  | (White Noise Removed)|     | (Impulse Removed)  |     | (Periodic Removed)  |     | (Drift
Removed)       |
  +---------------------+     +---------------------+     +---------------------+     +------------------------+
```

DSP Block Diagram



Code:

```
fs = 1000;
t = 0:1/fs:1;
signal = sin(2*pi*50*t);
% White Gaussian Noise
white_noise = 0.5 * randn(size(t));
noisy_white = signal + white_noise;
filtered_white = lowpass(noisy_white, 100, fs);
% Impulse Noise
impulse_noisy = imnoise(signal, 'salt & pepper', 0.02);
filtered_impulse = medfilt1(impulse_noisy, 3);
% Periodic Noise
periodic_noisy = signal + sin(2*pi*60*t);
notchFilter = designfilt('bandstopiir', 'FilterOrder', 2,...
   'HalfPowerFrequency1', 59,...
   'HalfPowerFrequency2', 61, 'DesignMethod', 'butter', 'SampleRate', fs);
filtered_periodic = filtfilt(notchFilter, periodic_noisy);
% Baseline Wander
baseline_wander = lowpass(signal, 0.1, fs);
highpass_filtered = highpass(baseline_wander, 0.1, fs);
% Plot results
figure;
subplot(4,2,1);
plot(t, signal);
title('Original Signal');
subplot(4,2,2);
plot(t, noisy_white);
title('Noisy Signal (White Noise)');
subplot(4,2,3);
plot(t, filtered_white);
title('Filtered (White Noise)');
subplot(4,2,4);
plot(t, impulse_noisy);
```

```
title('Signal with Impulse Noise');
subplot(4,2,5);
plot(t, filtered_impulse);
title('Filtered (Impulse Noise)');
subplot(4,2,6);
plot(t, periodic_noisy);
title('Signal with Periodic Noise');
subplot(4,2,7);
plot(t, filtered_periodic);
title('Filtered (Periodic Noise)');
subplot(4,2,8);
plot(t, baseline_wander, t, highpass_filtered);
title('Baseline Wander & High-pass Filtered');
legend('Before Filtering', 'After Filtering');
```

2. Removal of noise from Image data for further Image processing.

Block Diagrams

1. Averaging Filter Process

```
Input Image --> Add Salt-and-Pepper Noise --> Apply Averaging Filter
--> Denoised Image
```

Steps:

1. Input image is loaded.
2. Salt-and-pepper noise is added.
3. Convolution with an averaging filter kernel is performed.
4. The resulting image is displayed as the output.

---

2. Median Filter Process

mathematica

Copy code

```
Input Image --> Add Salt-and-Pepper Noise --> Apply Median Filter -->
Denoised Image
```

Steps:

1. Input image is converted to grayscale (if required).
2. Salt-and-pepper noise is added to the image.
3. A median filter is applied to a local neighborhood of each pixel.
4. The resulting image is displayed as the output.

Code:

Filter the noise image with an averaging filter:

```
a=imread('eight.tif');
imshow(a);
j=imnoise(a,'salt & pepper',0.02);
figure,imshow(j);
h=fspecial('average',3);
k=filter2(h,j)/255; %convolution
figure,imshow(k);
```

Median filters to filter the noisy image:

```
% Read the color image
a = imread('kku.jpg');
% Convert to grayscale
grayImage = rgb2gray(a);
% Add salt-and-pepper noise to the grayscale image
j = imnoise(grayImage, 'salt & pepper', 0.02);
% Apply median filtering
k = medfilt2(j, [3 3]);
% Display results
subplot(1, 3, 1);
imshow(grayImage);
title('Original Grayscale Image');
subplot(1, 3, 2);
imshow(j);
title('Noisy Grayscale Image');
subplot(1, 3, 3);
imshow(k);
title('Denoised Grayscale Image (Median Filter)');
```

3. Design of FIR adaptive filter as an Equalization filter

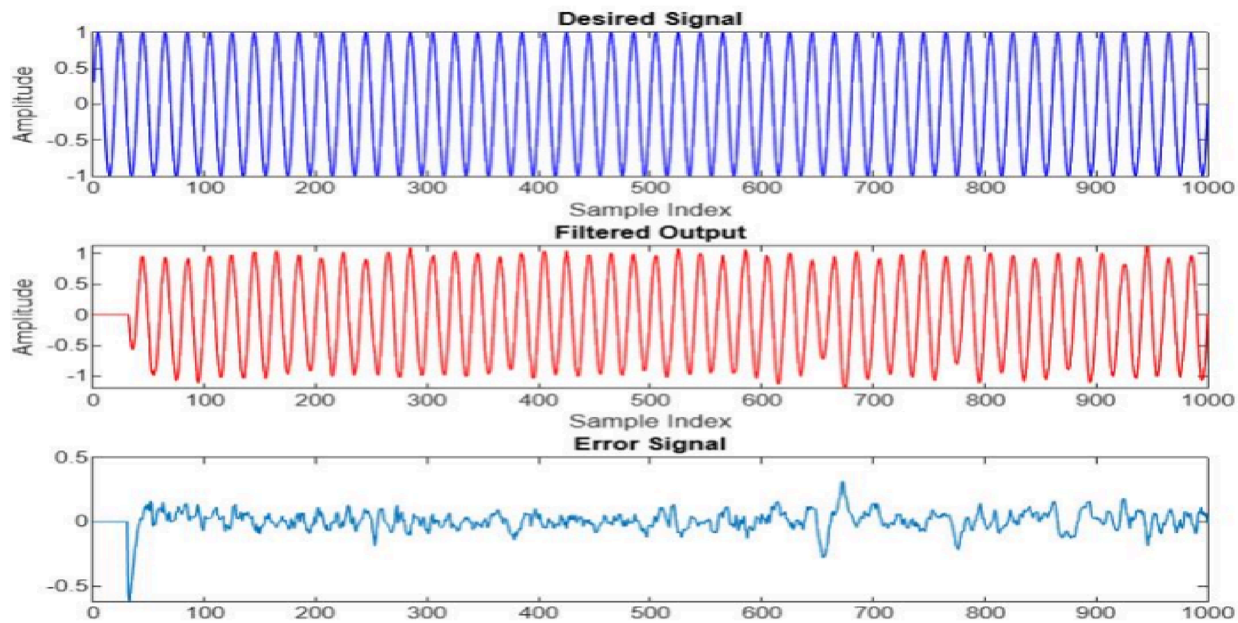
```
  +------------------------+
             |     Input Signal x[n]   |
             +------------------------+
                        |
                        v
             +------------------------+
             |  Delay Line (Tap Delay) |
             |  Generates x_vec[n]     |
             +------------------------+
                        |
                        v
             +------------------------------+
             | FIR Filter with Adjustable    |
             | Coefficients (h[n])           |
             | y[n] = h[n] * x_vec[n]        |
             +------------------------------+
                        |
                        v
             +---------------------+
+------------------------+
             |  Filtered Output y[n]       |  Desired Signal d[n]
|
             +---------------------+
+------------------------+
                        |                            |
                        v                            |
             +------------------------------+        |
             | Error Signal e[n] = d[n] - y[n]        |
             +------------------------------+        |
                        |                            |
                        v                            |
             +----------------------------------------+
             | Coefficient Update (LMS Algorithm)      |
             | h[n+1] = h[n] + 2μ e[n] x_vec[n]        |
             +----------------------------------------+
```

```
              |
              v
+-------------------------+
| Updated Coefficients h[n]|
+-------------------------+
```

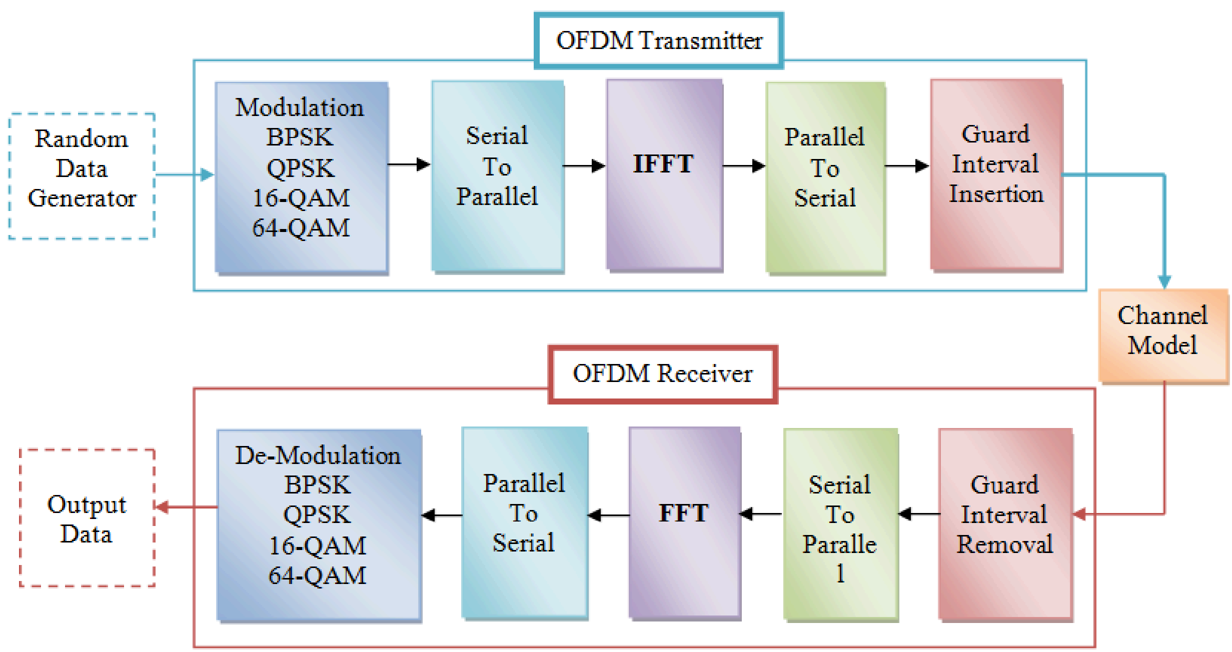
Code:

```
clc
clear all
close all
% Parameters
N = 32;
mu = 0.01;
nIterations = 1000;
x = sin(2*pi*0.05*(0:nIterations-1)) + 0.5*randn(1, nIterations);
d = sin(2*pi*0.05*(1:nIterations));
h = zeros(1, N);
for n = N:nIterations
x_vec = x(n:-1:n-N+1);
y(n) = h * x_vec';
e(n) = d(n) - y(n);
h = h + 2 * mu * e(n) * x_vec;
end
figure;
subplot(3,1,1);
plot(d, 'b');
title('Desired Signal');
xlabel('Sample Index');
ylabel('Amplitude');
subplot(3,1,2)
plot(y, 'r');
title('Filtered Output');
xlabel('Sample Index');
ylabel('Amplitude');
subplot(3,1,3);
plot(e);
title('Error Signal');
```

OUTPUT:



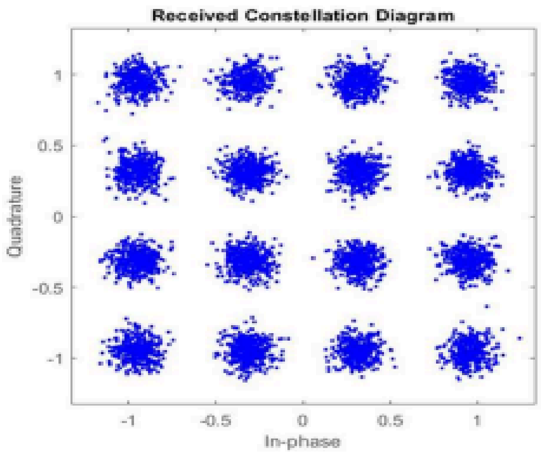
4. Develop OFDM standard complaint waveforms using FFT and other Blocks

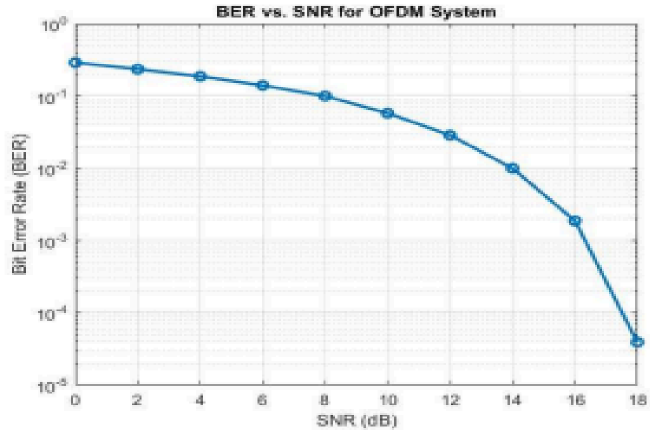


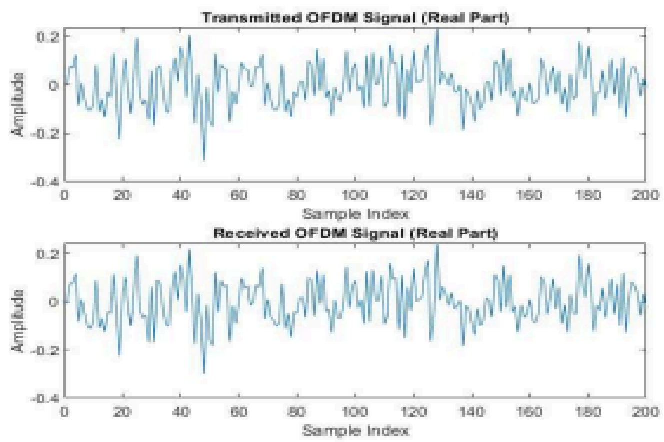

```
N = 64; % Number of subcarriers
M = 16; % Modulation order (16-QAM)
numSymbols = 100; % Number of OFDM symbols
cp_len = 16; % Cyclic prefix length
```

```
snrRange = 0:2:20; % SNR range in dB
berValues = zeros(size(snrRange)); % Preallocate for BER values
bits = randi([0 1], numSymbols * N * log2(M), 1); % Generate random bits
qamSymbols = qammod(bits, M, 'InputType', 'bit', 'UnitAveragePower', true);
ofdmSymbols = reshape(qamSymbols, N, numSymbols);
ifftSymbols = ifft(ofdmSymbols, N);
cpSymbols = [ifftSymbols(end-cp_len+1:end, :); ifftSymbols];
txSignal = cpSymbols(:);
% Loop through SNR values to calculate BER
for i = 1:length(snrRange)
snr = snrRange(i); % Current SNR value
rxSignal = awgn(txSignal, snr, 'measured'); % Add noise
rxSymbols = reshape(rxSignal, N + cp_len, numSymbols);
rxSymbols = rxSymbols(cp_len+1:end, :);
fftSymbols = fft(rxSymbols, N);
rxBits = qamdemod(fftSymbols(:), M, 'OutputType', 'bit', 'UnitAveragePower',
true);
[numErrors, berValues(i)] = biterr(bits, rxBits); % Calculate BER
end
% Plot BER vs SNR
figure;
semilogy(snrRange, berValues, '-o', 'LineWidth', 2);
grid on;
title('BER vs. SNR for OFDM System');
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
% Other plots for transmitted and received signals
figure;
subplot(2,1,1);
plot(real(txSignal(1:200)));
title('Transmitted OFDM Signal (Real Part)');
xlabel('Sample Index');
ylabel('Amplitude');
subplot(2,1,2);
plot(real(rxSignal(1:200)));
title('Received OFDM Signal (Real Part)');
xlabel('Sample Index');
ylabel('Amplitude');
% Constellation diagram
scatterplot(fftSymbols(:));
title('Received Constellation Diagram');
xlabel('In-phase');
ylabel('Quadrature');
```

Output:







5. Digital modulation and demodulation in AWGN and other channel setting.

```
+------------------------+
|   Random Data Input    |
+------------------------+
           |
           v
+------------------------+
|    QPSK Modulation     |
+------------------------+
           |
           v
+------------------------+
|    IFFT (OFDM Signal)  |
+------------------------+
           |
           v
```

```
+------------------------+
| Add Cyclic Prefix (CP)|
+------------------------+
           |
           v
+------------------------+
| Transmit Through AWGN |
|       Channel          |
+------------------------+
           |
           v
+------------------------+
| Remove Cyclic Prefix  |
+------------------------+
           |
           v
+------------------------+
|        FFT             |
+------------------------+
           |
           v
+------------------------+
|   QPSK Demodulation    |
+------------------------+
           |
           v
+------------------------+
| Compare Data & Compute|
|       Bit Errors       |
+------------------------+
```

CODE:

```
N = 4; % Number of subcarriers
Cp_len = 2; % Length of cyclic prefix
mod_order = 4; % QPSK (Quadrature Phase Shift Keying), 4 symbols
SNR = 20; % Signal-to-Noise Ratio in dB
data = randi([0 mod_order-1], N, 1);
modulated_data = qpsk_mod(data);
ofdm_time_domain = ifft(modulated_data, N);
ofdm_with_cp = [ofdm_time_domain(end-Cp_len+1:end); ofdm_time_domain];
received_signal = awgn(ofdm_with_cp, SNR, 'measured');
received_signal_no_cp = received_signal(Cp_len+1:end);
received_modulated_data = fft(received_signal_no_cp, N);
demodulated_data = qpsk_demod(received_modulated_data);
disp('Original Data:');
disp(data);
subplot(2,2,1);
plot(data);
disp('Demodulated Data:');
```

```
disp(demodulated_data);
subplot(2,2,2);
plot(demodulated_data);
disp(['Bit Error Rate: ', num2str(sum(data ~= demodulated_data) / N)]);
function qpsk_symbols = qpsk_mod(data)
qpsk_symbols = exp(1i * (pi/2) * (data + 1)); % Mapping [0,1,2,3] -> QPSK
end
function data = qpsk_demod(received_symbols)
phase_angles = angle(received_symbols);
data = mod(round((phase_angles + pi/4) / (pi/2)), 4); % Demapping
end
```

OUTPUT:

```
Original Data:
     3
     2
     2
     2

Demodulated Data:
     1
     0
     3
     3

Bit Error Rate: 1
```

6. Compress image using different standards such as JPEG, JPEG 2000 and SPIHT and EZW codes

```
        +---------------------+
        |   Original Image    |
        |  (cameraman.tif)    |
        +---------------------+
                  |
                  v
+------------------------------------+
|        Compression Methods         |
|                                    |
| 1. JPEG (Quality=50)               |
| 2. JPEG 2000 (CompressionRatio=20)|
| 3. SPIHT (maxloop=10)              |
| 4. EZW (maxloop=10)                |
+------------------------------------+
                  |
                  v
+------------------------------------+
|        Decompressed Images         |
|                                    |
| 1. JPEG Compressed Image           |
| 2. JPEG 2000 Compressed Image      |
| 3. SPIHT Decompressed Image        |
| 4. EZW Decompressed Image          |
+------------------------------------+
                  |
                  v
+------------------------------------+
|        PSNR Calculation            |
|                                    |
| PSNR for:                          |
| 1. JPEG                            |
| 2. JPEG 2000                       |
| 3. SPIHT                           |
| 4. EZW                             |
+------------------------------------+
                  |
                  v
+------------------------------------+
|          Display Results           |
|  - Images                          |
|  - PSNR Values in dB               |
+------------------------------------+
```

Code:

```
clc;
close all;
img = imread('cameraman.tif');
if size(img, 3) == 3
img = rgb2gray(img);
end
figure, imshow(img), title('Original Image');
imwrite(img, 'compressed_jpeg.jpg', 'Quality', 50);
jpeg_img = imread('compressed_jpeg.jpg');
figure, imshow(jpeg_img), title('JPEG Compressed Image');
imwrite(img, 'compressed_jpeg2000.jp2', 'CompressionRatio', 20);
jpeg2000_img = imread('compressed_jpeg2000.jp2');
figure, imshow(jpeg2000_img), title('JPEG 2000 Compressed Image');
wcompress('c', img, 'cameraman_spiht.wtc', 'spiht', 'maxloop', 10); % Compress
using SPIHT
decompressed_spiht = wcompress('u', 'cameraman_spiht.wtc'); % Decompress
figure, imshow(uint8(decompressed_spiht)), title('SPIHT Compressed Image');
wcompress('c', img, 'cameraman_ezw.wtc', 'ezw', 'maxloop', 10); % Compress
using EZW
decompressed_ezw = wcompress('u', 'cameraman_ezw.wtc'); % Decompress
figure, imshow(uint8(decompressed_ezw)), title('EZW Compressed Image');
original = double(img);
jpeg_psnr = psnr(double(jpeg_img), original);
jpeg2000_psnr = psnr(double(jpeg2000_img), original);
spiht_psnr = psnr(double(decompressed_spiht), original);
ezw_psnr = psnr(double(decompressed_ezw), original);
fprintf('PSNR Values:\n');
fprintf('JPEG: %.2f dB\n', jpeg_psnr);
fprintf('JPEG 2000: %.2f dB\n', jpeg2000_psnr);
fprintf('SPIHT: %.2f dB\n', spiht_psnr);
fprintf('EZW: %.2f dB\n', ezw_psnr);
```