APPENDIX 1

MATLAB PROGRAMME FOR MB OFDM
CHANNEL MODEL

UWB Channel Model

clear; % Program to generate channel co-efficients for N runs.
close all;
clear all;

cm_num=input('Enter Channel model Number');

num_channels = 100; % Number Of channels

ts=0.167; % Sampling time period

if cm_num == 1, % From Molisch paper
    Lam = 0.0233; lambda = 2.5;
    Gam = 7.1; gamma = 4.3;
    std_ln_1 = 4.8 / sqrt(2);
    std_ln_2 = 4.8 / sqrt(2);
    nlos = 0;
end
\texttt{std\_shdw} = 3;

\textbf{elseif} cm\_num == 2,

\texttt{Lam} = 0.4; \texttt{lambda} = 0.5;

\texttt{Gam} = 5.5; \texttt{gamma} = 6.7;

\texttt{std\_ln\_1} = 4.8 / \texttt{sqrt(2)};

\texttt{std\_ln\_2} = 4.8 / \texttt{sqrt(2)};

\texttt{nlos} = 1;

\texttt{std\_shdw} = 3;

\textbf{elseif} cm\_num == 3,

\texttt{Lam} = 0.0667; \texttt{lambda} = 2.1;

\texttt{Gam} = 14.00; \texttt{gamma} = 7.9;

\texttt{std\_ln\_1} = 4.8 / \texttt{sqrt(2)};

\texttt{std\_ln\_2} = 4.8 / \texttt{sqrt(2)};

\texttt{nlos} = 1;

\texttt{std\_shdw} = 3;

\textbf{elseif} cm\_num == 4,

\texttt{Lam} = 0.0667; \texttt{lambda} = 2.1;

\texttt{Gam} = 24; \texttt{gamma} = 12;

\texttt{std\_ln\_1} = 4.8 / \texttt{sqrt(2)};

\texttt{std\_ln\_2} = 4.8 / \texttt{sqrt(2)};
nlos = 1;

std_shdw = 3;

end

std_L = 1/sqrt(2*Lam);

std_lam = 1/sqrt(2*lambda);

mu_const = (std_ln_1^2+std_ln_2^2)*log(10)/20;

h_len = 1000;

ngrow = 1000;

h = zeros(h_len,num_channels);

t = zeros(h_len,num_channels);

t0 = zeros(1,num_channels);

np = zeros(1,num_channels);

for k = 1:num_channels

tmp_h = zeros(size(h,1),1);

tmp_t = zeros(size(h,1),1);

if nlos == 1,

Tc = (std_L*randn)^2 + (std_L*randn)^2;

else

Tc = 0;

end
\( t_{0(k)} = T_c; \)

\( \text{path\_ix} = 0; \)

\begin{verbatim}
while (T_c < 10*Gam)
    Tr = 0;
    ln_xi = std\_ln\_1*randn;
    while (Tr < 10*gamma)
        t_val = (T_c+Tr);
        mu = (-10*Tc/Gam-10*Tr/gamma)/log(10) - mu_const;
        ln_beta = mu + std\_ln\_2*randn;
        pk = 2*round(rand)-1;
        h_val = pk * 10^((ln_xi+ln_beta)/20);
        path\_ix = path\_ix + 1;
    end
    if path\_ix > h_len,
        tmp_h = [tmp_h; zeros(ngrow,1)];
        tmp_t = [tmp_t; zeros(ngrow,1)];
        h = [h; zeros(ngrow,num\_channels)];
        t = [t; zeros(ngrow,num\_channels)];
        h\_len = h\_len + ngrow;
    end
    tmp_h(path\_ix) = h_val;
\end{verbatim}
tmp_t(path_ix) = t_val;

Tr = Tr + (std_lam*randn)^2 + (std_lam*randn)^2;

end

Tc = Tc + (std_L*randn)^2 + (std_L*randn)^2;

end

np(k) = path_ix;

[sort_tmp_t,sort_ix] = sort(tmp_t(1:np(k)));

t(1:np(k),k) = sort_tmp_t;

h(1:np(k),k) = tmp_h(sort_ix(1:np(k)));

fac = 10^(std_shdw*randn/20);

h(1:np(k),k) = h(1:np(k),k) * fac;

end

h_ct=h;

min_Nfs = 100;

N = max(1,ceil(min_Nfs*ts));

N = 2^nextpow2(N);

Nfs = N / ts;

t_max = max(t(:));

h_len = 1 + floor(t_max * Nfs);

hN = zeros(h_len,num_channels);
for k = 1:num_channels

    np_k = np(k);

    t_Nfs = 1 + floor(t(1:np_k,k) * Nfs);

    for n = 1:np_k

        hN(t_Nfs(n),k) = hN(t_Nfs(n),k) + h_ct(n,k);

    end

end

if N > 1,

    h = resample(hN, 1, N);

else

    h = hN;

end

h = h * N;

channel_energy = sum(abs(h).^2);

h_len = size(h,1);

t = [0:(h_len-1)] * ts;  \% for use in computing excess & RMS delays

excess_delay = zeros(1,num_channels);

RMS_delay = zeros(1,num_channels);

num_sig_paths = zeros(1,num_channels);
num_sig_e_paths = zeros(1,num_channels);

for k=1:num_channels

    sq_h = abs(h(:,k)).^2 / channel_energy(k);

    t_norm = t - t0(k);

    excess_delay(k) = t_norm * sq_h;

    RMS_delay(k) = sqrt((t_norm-excess_delay(k)).^2) * sq_h);

threshold_dB = -10;

temp_h = abs(h(:,k));

temp_thres = 10^(threshold_dB/20) * max(temp_h);

num_sig_paths(k) = sum(temp_h > temp_thres);

x = 0.85;

temp_sort = sort(temp_h.^2);

cum_energy = cumsum(temp_sort(end:-1:1));

index_e = min(find(cum_energy >= x * cum_energy(end)));

num_sig_e_paths(k) = index_e;

end

energy_mean = mean(10*log10(channel_energy));

energy_stddev = std(10*log10(channel_energy));

mean_excess_delay = mean(excess_delay)

mean_RMS_delay = mean(RMS_delay)
mean_sig_paths = mean(num_sig_paths)
mean_sig_e_paths = mean(num_sig_e_paths)

figure(1);

h_sum=sum(h,2)/num_channels;
stem(t,h_sum);
grid on;

%title('Impulse response of the UWB channel');
xlabel('Time in (nsec)');
axis([0 300 -0.2 0.2]);

figure(2);

temp_average_power = sum(h'.*(h)')/num_channels;
temp_average_power = temp_average_power/max(temp_average_power);

average_decay_profile_dB = 10*log10(temp_average_power);
plot(t,average_decay_profile_dB);
grid on;
axis([0 t(end) -60 0])

%title('Average Power Decay Profile');
xlabel('Delay (nsec)');
ylabel('Average power (dB)');
figure(3);

plot([1 num_channels],mean_RMS_delay*[1 1], 'k--', [1:num_channels], RMS_delay, 'm-');

%grid on;

title('RMS Delay');

xlabel('Channel Number');

ylabel('Delay in (ns)');

legend('Mean RMS Delay');

figure(4);

plot([1 num_channels],mean_excess_delay*[1 1], 'k--', [1:num_channels], excess_delay, 'm-');

%grid on;

title('Excess Delay');

xlabel('Channel Number');

ylabel('Delay in (ns)');

legend('Mean Excess Delay');

figure(5);

plot([1 num_channels],mean_sig_paths*[1 1], 'k--', [1:num_channels], num_sig_paths, 'm-');

%grid on;

title('Number of significant paths within 10 dB of peak')
xlabel('Channel number');

ylabel('No.of Paths');

legend('Mean no.of paths');

figure(6);

plot([1 num_channels], mean_sig_e_paths*[1 1], 'k--', [1:num_channels], num_sig_e_paths, 'm-');

grid on;

title('Number of significant paths capturing > 85% energy')

xlabel('Channel number');

ylabel('No.of Paths');

legend('Mean no.of paths');

figure(7);

figh = plot([1:num_channels], 10*log10(channel_energy), 'm-', ... [1 num_channels], energy_mean*[1 1], 'k-', ... [1 num_channels], energy_mean+energy_stddev*[1 1], 'b: ', ... [1 num_channels], energy_mean-energy_stddev*[1 1], 'b:');

xlabel('Channel number')

ylabel('dB')

title('Channel Energy');

legend(figh, 'Per-channel energy', 'Mean', '\pm Std. deviation', 0)