Section IV – Functions

Function: dialSequence.m

function x = dialSequence(digitarray,toneduration,pauseduration,fs)

output = 0;

for k = 1:length(digitarray);

tone = generateDTMF(digitarray(k),toneduration,fs);

silence = generateSilence(pauseduration,fs);

full = [tone, silence];

output = [output, full];

end;

x = output;

Function: direct\_fourier.m

function y = direct\_fourier(x , fs)

h = zeros(size(x));

n = 1;

for q = 1:fs

for k=1:fs

h(n) = h(n) + x(k)\*exp(-2\*pi\*j\*k\*n/(fs));

end

n = n+1;

end;

y = h;

Function: generate\_wave.m

function x = generate\_wave(amp,freq,samp,length)

T = (1/samp); % Sampling Period

n = (0:(length\*samp)); % Samples

t = n\*T; % Sampling Instances

x = amp\*cos(2\*pi\*freq\*t); % Signal

Function: generateDTMF.m

function x = generateDTMF(digit,duration,fs)

t = (0:1/fs:duration);

a = 0;

b = 0;

switch digit

case '1'

a = 697;

b = 1209;

case '2'

a = 697;

b = 1336;

case '3'

a = 697;

b = 1477;

case '4'

a = 770;

b = 1209;

case '5'

a = 770;

b = 1336;

case '6'

a = 770;

b = 1477;

case '7'

a = 852;

b = 1209;

case '8'

a = 852;

b = 1336;

case '9'

a = 852;

b = 1477;

case '0'

a = 941;

b = 1336;

case '\*'

a = 941;

b = 1209;

case '#'

a = 941;

b = 1477;

otherwise

a = 1;

b = 1;

end

x = cos(2\*pi\*a\*t)+cos(2\*pi\*b\*t);

Function: generateSilence.m

function x = generateSilence(duration,fs);

x = zeros(1,duration\*fs);

Function: plot\_power\_spectrum.m

function plot\_power\_spectrum(x,fs)

index = [1:(fs+1)];

figure('Color',[1 1 1]);

h = plot(index,abs(x.^2));

xlim([0 fs/2]);

box off;

set(h,'Linewidth',3);

set(h,'Color', [0.6 0.6 1]);

hold on;

box off;

xlabel('Physical Frequency (Hz)');

ylabel('Power');

Function: decodeDTMF.m

function [digit,starttime,endtime]=decodeDTMF(x1,fs)

% Normalize the signal

x1 = NormalizeSignal(x1);

% Find the signal's Start and End Points

indexstart = FindSignalStart(x1);

indexend = FindSignalEnd(x1);

% Create a "smaller" version of the original signal, with only the relevant

% part

subx1 = x1(indexstart:indexend);

% Let's get some power

y = fft(subx1);

% Plot confirms values obtained

plot\_power\_spectrum(y,length(y)-1);

% Because of the "mirroring", we need to omit frequency data above the

% halfway mark

cutoff = round(0.5\*length(y),0);

test = abs(y(1:cutoff));

% Create a threshold value so that the function doesn't report back very

% small values

threshold = (max(test)/2);

% Each Signal has two relevant frequencies: A 'low' and a 'high'

% These values are found, and then rounded (because my method of cutting

% the signal introduces a little error)

low = round(find(test>threshold,1)\*(fs/length(y)), -1);

high = round(find(test>threshold,1, 'last')\*(fs/length(y)), -1);

% Fortunately, all of the frequencies are more than 10 apart, so there

% shouldn't be any overlap in using this method of rounding.

% Every possible combination of addition for anticipated signals results in

% a unique number. Easier to build a case structure for.

combined = low+high;

%And a Hacky Case Structure to Output the correct string

switch combined

case 1910

digit = '1';

case 2040

digit = '2';

case 2180

digit = '3';

case 1980

digit = '4';

case 2110

digit = '5';

case 2250

digit = '6';

case 2060

digit = '7';

case 2190

digit = '8';

case 2330

digit = '9';

case 2150

digit = '\*';

case 2280

digit = '0';

case 2420

digit = '#';

otherwise

digit = 'UNKNOWN';

end

% Last bit is to report back the time the signal starts and ends.

starttime = indexstart/fs;

endtime = indexend/fs;

Function: NormalizeSignal.m

function y = NormalizeSignal(x)

x = x - mean(x); % Remove DC Bias

y = x/max(abs(x)); % This should work in theory.

% Guarantees less than 1, at least.

Function: FindSignalStart.m

function y = FindSignalStart(x)

threshold = (max(x)/2);

y = find(x>threshold,1);

Function: FindSignalEnd.m

function y = FindSignalEnd(x)

threshold = (max(x)/2);

y = find(x>threshold,1, 'last');