Building a Channel Sounder

Table of Contents

A **channel sounder** is a device that measures the channel response between a transmitter (TX) and receiver (RX). Channel sounders are key for researchers studying wireless channel propagation. They are also useful in bringing up any SDR platform to ensure the wideband response through the TX and RX chains is as expected. Indeed, after ensuring you can receive a tone correctly, use a channel sounder to make sure the device works and correctly characterize the TX and RX filters.

In going through this lab, you will learn to:

- Measure the channel frequency response via frequency-domain correlation
- Compute the time-domain channel impulse response via an IFFT
- Align the channel impulse response via peak detection
- Estimate the SNR
- Build a continuous monitor for the channel

Submissions: Run the lab with Pluto devices with either one or two hosts. Fill in all sections labeled `TODO` . Print and submit the PDF. Do not submit the source code.

We set the parameters

- `saveData`: Indicates if the data is to be saved
- `usePreRecorded`: Indicates if the data is to use the pre-save data. This way, you will not need to run the device

```
% Parameters
saveData = false;
usePreRecorded = false;
```

Creating the TX Waveform

In frequency-domain channel sounding, we create the waveform in frequency domain. First, we create a vector `xfd` of `nfft` QPSK symbols. Then, we take the IFFT to create the time-domain waveform.

```
nfft = 1024;
```

```
if usePreRecorded
    load txData;
else
    % TODO:  Create a vector, xfd, or nfft random QPSK symbols.
    %   xfd = ...


    % Save frequency-domain TX data
    if saveData
        save txData xfd;
    end
end

% TODO:  Take IFFT
%    x = ifft(xfd);
```

Recall from the previous lab that the TX must take values where `real(x)` and `imag(x)` are in `[-1,1]`. Re-scale x such that it has a 9 dB backoff from full scale, and then clip the signal so that it is in this range. Store the scaled and clipped signal in `xclip`.

```
% TODO:
%   xclip = ...
% Rescale and clip values for the backoff
```

Create the TX and RX objects

Following the initial lab, set the mode to the desired configuration (loopback, two hosts with single host, or two hosts with two hosts).

```
% TODO:  Set parameters
% Set to true for loopback, else set to false
loopback = false;

% Select to run TX and RX
runTx = true;
runRx = true;
```

Plug one or two Plutos into the USB ports of your computer. If it is correctly working, in each device the Ready light should be solid on and the LED1 light should be blinking. Next, run the following command that will detect the devices and create the TX and RX ob

```
% clear previous instances
clear tx rx

% add path to the common directory where the function is found
addpath('..\common');

% Parameters
fsamp = 30.72e6*2;  % Maximum sample rate
nsampsFrame = nfft;
```

```
% Run the creation function.  Skip this if we are using pre-recorded
% samples
if ~usePreRecorded
    [tx, rx] = plutoCreateTxRx(createTx = runTx, createRx = runRx, loopback = loopback, ...
        nsampsFrame = nsampsFrame, sampleRate = fsamp);
end
```

TX and RX Data via the Pluto Device

Now TX the clipped data `xclip` repreatedly through the Pluto device.

```
if runTx && ~usePreRecorded
    % TODO:  Use the tx.release and tx.transmitRelease commands to
    % continuously send xclip

end
% If not running the RX, stop the live script now
if ~runRx
    return;
end
```

On the RX Pluto device, receive the data

```
nbits = 12;               % number of ADC bits
fullScale = 2^(nbits-1);  % full scale

if ~usePreRecorded

    % TODO:  Capture data
    %   r = rx.capture(...)


    % TODO:  Scale to floating point
    %   r = ...

    % Save the data
    if saveData
        save rxDataSing r;
    end

else
    % Load pre-recorded data
    load rxDataSing;

end
```

Measuring the RX Channel Frequency Response

Since the signal `x` is being repeatedly transmitted, the channel will act as a circular convolution. Therefore, the channel frequency response can be estimated via:

`hfd(k) = rfd(k) / xfd(k)`

where `rfd = fft(r)`. Complete the first two sections in the function `estChanResp` to estimate the frequency response. You can ignore the other parts and outputs for now. After completing this section, run the following code.

```
% TODO:  Complete the TODO sections in estChanResp up to hfd = ...
% Then run:
hfd = estChanResp(r,xfd);
```

The vector `hfd` from the previous part provides the sampled channel frequency response. Plot the power gain in dB vs. frequency. The frequency should be in MHz and can be computed from the sample rate, `fsamp`. Use the command `fftshift` to place the DC frequency in the center.

You should see the following:

- The channel starts to roll off after about `|f| > 10 MHz`. This is not from the physical channel, but the filtering in the TX and RX filters. This is why the usable bandwidth for the device is about 20 MHz, sufficient for a 20 MHz 4G or 5G carrier as well as a 20 MHz 802.11 signal.
- There is no channel gain at DC. The lack of a DC component is since the Pluto has a DC filter to remove the DC component. This filtering is necessary in any direct conversion architecture since the LO image generally appears at DC.
- There is an arbitrary scaling since the RX has automatic gain control.

```
% TODO:  Compute the channel power gain and shift it using fftshift and
% plot against the frequency in MHz.  Label your aees.
%   hfddB = ...
%   fMHz = ...
%   plot(...)
```

Compute the Time-Domain Channel Impulse Response

We can compute the channel impulse response by simply taking the IFFT of the channel frequency response. This IFFT is performed in the function `estChanResp`. Complete the TODO section of that code and run the following command.

```
% TODO:  Complete the TODO sections in estChanResp up to h = ...
% Then run:
%     [hfd, h] = estChanResp(r,xfd);
[hfd,h] = estChanResp(r,xfd);
```

Plot the energy in each tap of `h` in dB using a `stem` plot. You should see a peak corresponding to the location of the main path. To make the plot attractive:

- The y-axis should be the energy per tap in dB. Note, the scale is arbitrary since the RX gain is not known.
- The x-axis should be the time in microseconds. Use the sample rate.
- Set the `BaseValue` to 10 dB below `Emedian =` the median energy / tap.
- Use the `ylim` command to set the y-axis range from the `Emedian-10` to `Emedian+snrMax` where `snrMax` is defined below.
- Label the axes

```
% Maximum range used for scaling the plot.
snrMax = 60;

%  TODO:
%     tus = ...  % Sample time in micro-seconds
%     hpow = ...
%     Emedian = ...
%     stem(tus, hpow, 'BaseValue', ...);
%     ylim([...]);
```

The peak sample can be in an arbitrary position since it depends on when the RX sampled the data. Therefore, it is generally better to align the peak so that it is in a fixed position. Complete the code in `estChanResp` that aligns the peak so that `|h(k)|` is maximum when `k=opt.nleft+1.` Then, there will be a small number, `opt.nleft` samples to the left of the peak.

```
% TODO:  Complete the code estChanResp for the peak alignment.
% Then run the following code.
nleft = 8;
[~,h] = estChanResp(r,xfd,'nleft',nleft);
```

In order that you can see the channel around the peak well, plot the first `nplot=64` samples of `h(k), k=1,...,nplot.` Since the peak is at `k=nleft+1,` we say that sample `k` is at a *relative delay* of

`t(k) = (k-nleft-1)/fsamp`

Plot the energy in each `h(k)` in dB vs. the relative delay in micro-seconds. Again, most of the samples are from the filtering and the fact that the actual main path arrives between two samples. To see the multi-path in a room, we would need a higher sample rate.

```
nplot = 64;

% TODO:  Plot the energy / tap vs. relative delay
```

Estimating the SNR

We can now estimate the SNR as follows. Each tap `h(k)` is the sum of signal and noise:

`h(k) = s(k) + w(k)`

where `s(k)` is the signal component and `w(k)` is the noise component. Now, recall that peak is aligned at `k=nleft+1`. Since the channel has a limited delay spread, the signal component should be small outside a small number of taps to the left and right of the peak. That is,

```
s(k) ~= 0   for k > nsig = nleft+nright+1,
```

where `nleft` and `nright` are the maximum number of samples to the left and right of the signal where we expect signal energy. In these samples, `h(k)` is just noise so we can estimate the noise energy per sample via the average of the noise on these samples.

```
Enoise = mean( abs(h(nsig:end).^2 )
```

The samples `h(k)`, `k=1,...,nsig` will contain the total signal energy + noise energy for `nsig` taps. Hence, we have:

```
sum( abs(h(1:nsig).^2 ) ~= Esig + nsig*Enoise
```

Using this equation, we can estimate the signal energy as:

```
Esig = sum( abs(h(1:nsig).^2 )  - nsig*Enoise
```

Since the measurements have random errors, you can get a negative number sometimes. So, you will want to limit this value to a small positive number. Then the snr can be estimated via

```
snr = pow2db( Esig / Enoise )
```

Complete the code in `estChanResp` for computing the SNR. Then, run the following command to print the SNR.

```
% TODO:  Complete the code in estChanResp for computing the SNR
% and then run the following code to print the SNR.
nright = 8;
[~,~,snr] = estChanResp(r,xfd,'nleft',nleft,'nright',nright);
fprintf(1,'Snr = %7.2f\n', snr);
```

Finally, it is useful to re-scale the time-domain taps by the estimated noise so that:

```
hscaled = h / sqrt(Enoise)
```

In this way, `abs(h(k))^2` is the energy in the tap relative to the noise and hence represents the SNR per tap.

Complete the code `estChanResp` for scaling the energy per tap. Then, replot the SNR/tap vs. relative delay in micro-seconds. Label and set the axes as before.

```
% TODO:  Complete the code in estChanResp for normalize to the noise
[~,h] = estChanResp(r,xfd,'nleft',nleft,'nright',nright, 'normToNoise',true);

% TODO:  Plot the SNR / tap vs. relative delay
```

Create a Continuous Monitor

We will now continuously monitor and plot the channel impulse response and SNR.

```
if runTx && ~usePreRecorded
    % TODO:  Use the tx.release and tx.transmitRelease commands to
    % continuously send xclip
    tx.release();
    tx.transmitRepeat(xclip);
end

% Number of captures
ncaptures = 100;

% Initialize SNR vector
snr = zeros(ncaptures,1);

% Load pre-recorded data if required
if usePreRecorded
    load rxDatMult;
    ncaptures = size(rdat,2);
else
    rdat = zeros(nfft, ncaptures);
end


if usePreRecorded
    r = rdat(:,1);
else
    % TODO:  Capture initial data
    %    r = ...
end

% Get the channel response and initial SNR
[~,h,snr0] = estChanResp(r,xfd,'nleft',nleft,'nright',nright, 'normToNoise',true);

subplot(1,2,1);

% TODO:  In the first sub-plot, plot the SNR/tap vs. relative delay as
% before.  When you call the stem plot get a handle
%
%    tus1 = relative delays in micro-seconds for the first nplot samples
%    hpow1 = vector of SNRs / tap for the frst nplot samples
%    p = stem(tus1, hpow1, ...)
%
% Label the axes and set the limits correcly


% In the second sub-plot, we will plot the SNR vs capture
subplot(1,2,2);
psnr = plot((1:ncaptures), snr);
xlim([1,ncaptures]);
ylim([-5,snrMax]);
grid on;
ylabel('SNR [dB]');
xlabel('Capture');
```

```
% Set pointers to Y data for both plots
p.YDataSource = 'hpow1';
psnr.YDataSource = 'snr';



snr = zeros(ncaptures,1);
for t = 1:ncaptures
    if ~usePreRecorded

        % TODO:  Capture data
        %    r = rx.capture(...);
        r = rx.capture(nsampsFrame);
        r = single(r)/fullScale;

        % Save data if required
        if saveData
            rdat(:,t) = r;
        end
    else
        % Load pre-recorded
        r = rdat(:,t);
    end


    % Re-compute the impulse response
    [~,h,snr(t)] = estChanResp(r,xfd,'nleft',nleft,'nright',nright, ...
        'normToNoise',true);

    % TODO:  Re-compute hpow1 = SNR/tap on the first nplot samples.


    % Redraw plot
    refreshdata
    drawnow

end

% Save data
if saveData
    save rxDatMult rdat;
end
```

Further Enhancements

There are several ways we can improve the channel sounder. Feel free to take this code and modify it if you are interested.

- We can integrate over multiple RX frames to average out the noise.
- Also, if we have multiple RX frames, we can estimate the carrier frequency offset (CFO). Estimating and correcting the CFO is essential to integrate over multiple frames.

- Multiple frames will also allow us to measure fading, which is the variation of the channel over time. This process is discussed in the wireless communications class
- We can use a wider band sounder (i.e., higher sample rate) to resolve closer multipath component
- A very similar method can be used for RADAR