Orthogonal Frequency-Division Multiplexing

Victoria Preston and Emily Tumang
Principles of Wireless Communications

***Abstract*—We simulated an orthogonal frequency-division multiplexing (OFDM) system using MatLab by 'transmitting' a packet structured similarly to the wi-fi standard through a non-flat channel with frequency offset, applying the Schmidl-Cox algorithm to the 'received' data, and leveraging the flat fading assumption to estimate the channel in order to decode a 15-symbol payload. Notable factors influencing signal success included the number of channel estimates performed and precision of indexing within a packet.**

I. Lab Summary and Provided Resources

The objective of this lab was to simulate an orthogonal frequency-division multiplexing (OFDM) system. Provided to the students were some fundamental MatLab functions which allowed for a non-flat channel simulation, a frequency-offset simulation, and implemented a Schmidl-Cox algorithm.

II. Theoretical Overview

OFDM is a method of transmitting information over a non-flat channel in order to increase available bandwidth. To do so, data is transmitted in the frequency domain over smaller subchannels, each of which is individually/independently small enough to be considered flat. Transmission frequencies are chosen such that transmitted pulses are orthogonal to each other in the frequency domain, eliminating inter-symbol interference.

III. Implementation and Results

For this lab, we primarily used MatLab to simulate the transmission and reception of the data vectors. Provided more time to troubleshoot the USRP devices, we would have also used GNU-Radio and Python in order to physically transmit and receive signals.

A. Transmission

In order to transmit OFDM signals, we generated a data vector with a structure similar to that of a wi-fi packet. The first section of this consisted of a header of 64 random BPSK values, repeated three times. This was used to determine the start of the packet and for frequency shift correction. Second was a set of 10 OFDM symbols known to both the receiver and the transmitter, used for channel estimation. Each symbol consists of the IFFT of 64 samples, with the last 16 samples, called a cyclic prefix, appendd to the beginning of the symbol for a total of 80 samples. Finally, 15 unknown symbols were formatted in the same was as the known data and transmitted; this is the data payload.

The data vector was then transmitted through a non-flat channel with frequency shift. The final signal is shown in Figure 1.

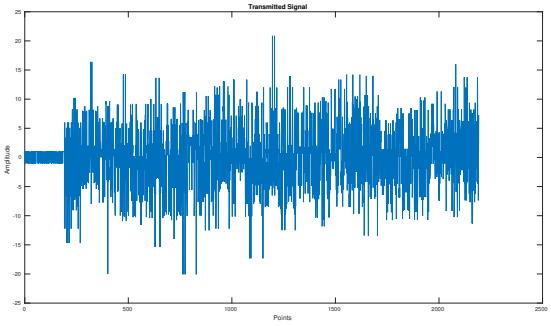

Fig. 1: The transmitted signal. Notice that the header is clearly encoded in the time domain, whereas the rest of the data is encoded in the frequency domain.

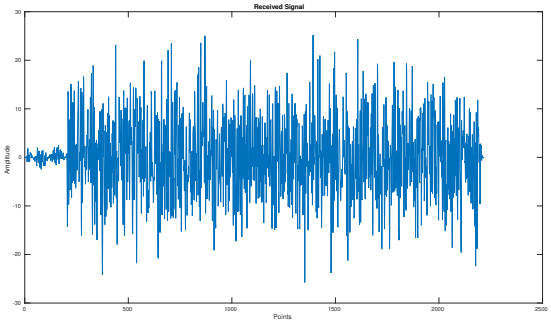

Fig. 2: The received signal with noise and frequency offset applied.

B. Reception

The signal is transformed in a MatLab function which simulates the effects of a non-flat channel with noise and frequency error, producing the signal shown in Figure 2.

1) Frequency Correction: We used the Schmidl-Cox algorithm (which was provided through lab sample code) to correct frequency errors between the transmitted and received signal, as implemented in the provided correct_cfo_schmidl_cox.m file. This works by detecting the beginning of the second of the three identical header vectors (the first acts only as a buffer, and is mostly removed on reception) and comparing the angle between each point between the two vectors to estimate the frequency offset. This is ultimately used to correct the rest of the received signal.

2) Channel Estimation: In order to estimate the channel, the cyclic prefix of each known transmitted packet is removed and the data is converted to the frequency domain via FFT. It is

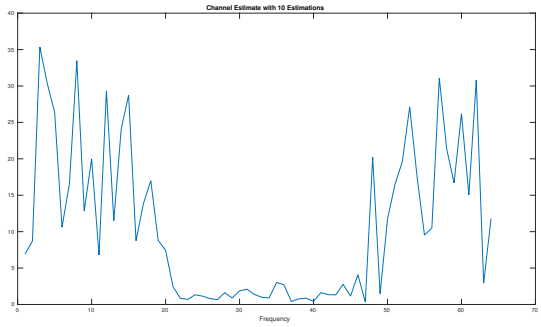

Fig. 3: The estimated channel. Notice the low-pass nature of the channel.

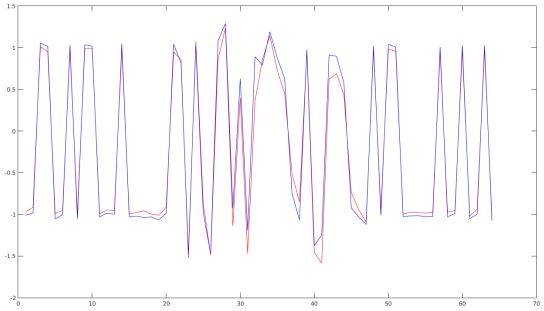

Fig. 4: A decoded payload symbol demonstrating the effects of the number of channel estimations on the resulting data. The blue line shows the signal with 3 estimations, and the red one with 10.

then divided by the original transmission, with the dividend assumed to be the channel characteristic (since each subchannel is assumed flat, H can be assumed to be multiplicative). Over multiple symbol channel estimates the results are averaged together to produce a more accurate/holistic channel estimate. The channel that was estimated in our work is illustrated in Figure 3.

3) Symbol Processing: Finally, each payload symbol is stripped of its cyclic prefix, converted to the frequency domain (via FFT) and divided by the channel estimate in order to recover the original data. The results of this process are shown in Figures 4 and 5 .

IV. Conclusion and Extension

We were able to simulate an OFDM system, as shown by the results in the previous section. As shown in Figure 4, the number of channel estimations strongly affects the quality of the estimation and therefore the quality of the decoded data. For this reason, we chose to deviate from the wi-fi standard and use as few as 10 and as many as 20 known packets for channel estimation, as opposed to the usual 3. We also did not use pilot tones, in part because of some frustrations in implementing what we have demonstrated, and in part that it did not fulfill some of our learning goals for the lab.

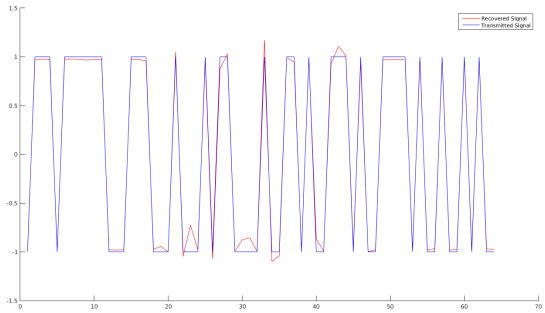

Fig. 5: A decoded payload symbol in comparison to the true transmitted signal. We had great results in this recovery.

One factor that strongly affected decoding was indexing; we noticed that if we began any part of the decoding at the wrong point in the signal, the result would be hopelessly garbled. In this way, correct estimation and data robustness is absolutely critical in this method. We wonder how this is addressed in the real world when packets can be disrupted, dropped, and so forth. Some amount of redundancy in the system is built perhaps? We would be interested in learning more on this matter.

V. Appendix

Our transmission and reception code, in addition to our channel estimation is included. For all of our code, you can find our repo https://github.com/vpreston/ofdm$_o linpwchere$.

A. Transmission

1) transmitter.m:

```
% This code creates a transmission signal that can
    be sent with an ESRP
% software defined radio unit.

% This code creates a random signal vector using the
     helper function create_vector, performs an IFFT
    , identifies a
% cyclic prefix, then appends the prefix to the
    front of the transformed
% random signal vector.


% establish the header which will be in the time
    domain
header = create_header;

% make sure to save the signals for use by the
    receiver
known_signal = [header];
known = [];

% create the known data, 20 symbols
for j = 1:20
    data  = create_vector(64);
    signal = ifft(data)*64;
    cyclic_prefix = signal(end-15:end);
    known_signal = [known_signal;cyclic_prefix;
    signal];
    known = [known;cyclic_prefix;data];
end

final_signal = [known_signal];
```

```
% add the symbol payload of between 10-20 symbols
for i = 1:15
    random_data = create_vector(64);
    random_signal = ifft(random_data)*64;
    random_cp = random_signal(end-15:end);
    true_data = [random_cp;random_signal];
    final_signal = [final_signal;true_data];
end

% transmit the information through the USRPs by
    creating file it can parse,
% in this case a .dat file
save('transmitter.dat', 'final_signal')
```

2) channel estimation:

```
function h = channel_estimate(signal,known)

    signal = signal(17:end);
    known = known(17:end);
    input = fft(signal);
    h = input./known;
end
```

2) create header:

```
function res = create_header()
    header = 2*round(rand(64,1)) - 1;
    header = repmat(header, [3 1]);
    res = header;
end
```

3) create vector:

```
function res = create_vector(l)
    signal = 2*round(rand(l,1)) - 1;
    res = signal;
end
```

B. Reception

1) reciever.m:

```

function receiver(y,known_signal)
    %create important indices
    training_start = 64*2 + 1;
    data_start = training_start  + 80*1;
    h = [];

    %find the start of the data and do a frequency
     correction
    signal = correct_cfo_schmidl_cox(y(50:end));
    known = known_signal;

    %perform the channel estimation
    for i=1:20
       h = [h,channel_estimate(signal(training_start
    +80*(i-1):training_start+79+80*(i-1)),known
    (1+80*(i-1):80+80*(i-1)))];
    end

    h = mean(h,2);

    %process the symbols
    symbols = signal(data_start:end);
   corrected_data = process_symbol(symbols(1:80),h);

   %plot everything
    hold on
    plot(real(corrected_data),'r-')
    plot(real(known(81+16:(81+79))),'b-')
    legend('Recovered Signal','Transmitted Signal')
end
```