Lab 2a - 2 I/O MIMO System

Pranavi Boyalakuntla
Jonathan Kelley

1. Overview

For this lab, we are implementing a receiver for a MIMO digital communication system with 2 transmitters and 2 receivers. In this lab, we will be working with data sent and recorded using USRP's. Our goal is to extract and decode the original data points from the received data by implementing various receiver schemes in the MATLAB development environment.

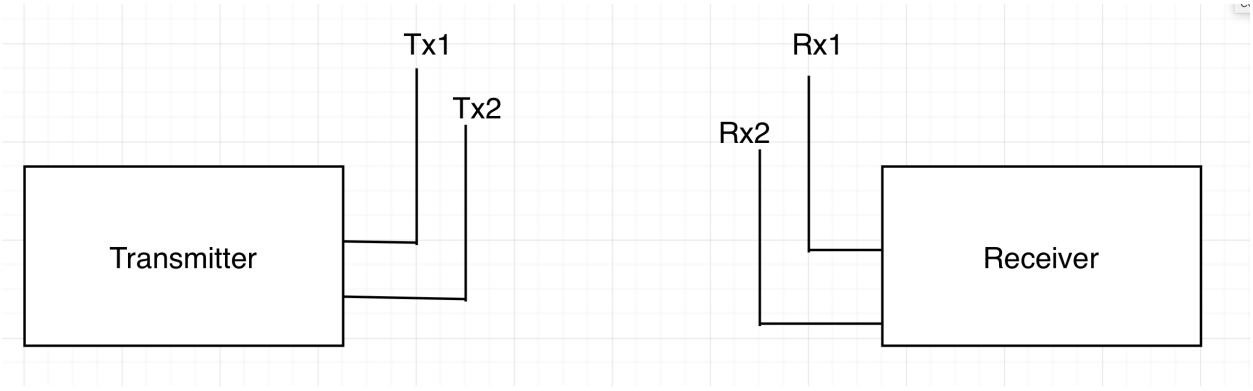

Figure 0: Potential set up for this data collection

The bits were transmitted using the following order.

1. 5000 zero samples from transmitter 1 and 2

2. Transmitter 1 sends 128 pseudo random bits, encoded using BPSK and 40 sample long rectangular pulses, transmitter 2 sends zeros.

3. 5000 zero samples from both transmitters

4. Transmitter 2 sends 128 pseudo random bits, encoded using BPSK and 40 sample long rectangular pulses, transmitter 1 sends zeros.

5. 5000 zero samples from both transmitters

6.Transmitters 1 and 2 send 1024 data bits (they are randomly generated here), using BPSK and rectangular pulses that are 40 samples each.

2. Receiver Design

2.0 Receiver Overview

In designing the receiver for our transmitter system, we first dove into the literature to gain a better understanding Multiple Input Multiple Output (MIMO), Space Division Multiple Access (SDMA), and channel estimation meant in the context of our system. We also made some assumptions to better constrain the final implementation. First, we assumed a flat-fading channel model as a result of ambiance white gaussian noise. Next, we assumed that the transmitted signal bandwidth with significantly less than the carrier bandwidth. We also assume that the carrier frequencies are synchronized such that the channels do not change.

Our research into the MIMO system led us to a matrix representation of the system. For any transmitted signal, we would have two corresponding received signals: one for either antenna. Let's denote the signals transmitted as $Tx_1 = x_1$ and $Tx_2 = x_2$. Let's denote the signals received by the receivers as $Rx_1 = y_1$ and $Rx_2 = y_2$.

The challenge with processing the received data is processing out the data that this receiver is not meant to capture. Both of the receivers will hear a combination of both transmitted messages weighted by the channel they were sent through and some additional noise represented by n[k].

$$y_1[k] = h_{11}x_1[k] + h_{12}x_2[k] + n_1[k] \qquad y_2[k] = h_{21}x_1[k] + h_{22}x_2[k] + n_2[k]$$

In both antennae, noise would play a significant factor in the final received signal. To stay inline with the literature and represent our system as a linear equation, we can combine our x, y, and H terms.

$$y = Hx + n \tag{1}$$

Correct for Timing Delays

We know that the carrier frequencies of the transmitter and receiver are synchronized such that we do not need to account for any frequency or phase changes that occur. However, we still need to account for timing delays. In order to do this, we can take the cross correlation between the sent and received data. We can find the lag at which the greatest correlation occurs and consider that or delay. This was 20 samples.

Channel Estimation

There are four channels we need to estimate.

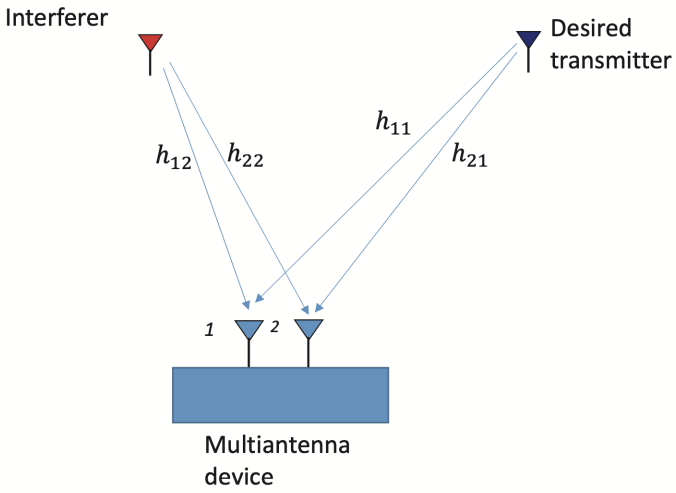

Figure 1: Breakdown of how the four channels might be positioned.

We can utilize the two portions of the signal where only one transmitter is transmitting to accurately estimate the channel between each transmitter and both receivers. For example, while just transmitter 1 is transmitting, we can estimate the h11 and h21 channels as shown in figure 1.

Once we have isolated these portions, we can divide the sent signal by the received signal to determine the channel properties.

$$h = y./x \tag{2}$$

As shown in equation 1, we require an H matrix which contains all the channel estimations in it.

$$H = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \tag{3}$$

2.1 Zero-Forcing Receiver

For the first receiver, we can implement a a "zero-forcing" receiver. Here, we can use matrix math to compensate for the channel overlap and try to intelligently cancel out noise from channels we are not interested in listening to. Using the equation we derived before, we can combine these "listening weights" with the received and sent signal relationship.

We know that if we apply some weight vector w to y, that we could get an accurate estimation of s.

$$\hat{x} = w^H y \tag{4}$$

We also know that (from equation 1) that y is equal to the sent signal weighted by the channel.

$$\hat{x} = w^H H x \tag{5}$$

Eqn 5. shows how the estimate of x ($\hat{x}$) would be equal to the product of the received signal and some weight vector if we assumed that noise was negligible. This weight vector would let us filter out our total received signal for either receiver, cancelling out any interference effects. In other words, we can set the weighted channels between the receiver the transmitter we are trying to block out to 0.

For the first transmitter, this would look like:

$$w_1^* h_{11} + w_2^* h_{21} = 1 \qquad (5) \qquad\qquad w_1^* h_{12} + w_2^* h_{22} = 0 \qquad (6)$$

For the second transmitter, this would look like:

$$w_1^* h_{11} + w_2^* h_{21} = 0 \qquad (7) \qquad\qquad w_1^* h_{12} + w_2^* h_{22} = 1 \qquad (8)$$

Represented completely in matrix form, this would be:

$$\hat{x}_1 = w_1^H \boldsymbol{y} \qquad (9)$$

$$\hat{x}_1 = w_2^H \boldsymbol{y} \qquad (9)$$

Implemented in MATLAB, this would look like:

```
% we know that H* x w = [1; 0] for x1
w1 = inv(H') * [1; 0];
w2 = inv(H') * [0; 1];

x1_hat = w1' * y;
x2_hat = w2' * y;
```

Because we cancel out the alternate channels with the weight matrix, this receiver type is known as a zero-forcing receiver.

2.2 MMSE Detector

For the first receiver, we be implemented a zero-forcing receiver. Now, we'll implement a "minimum mean square error" receiver. This receiver technique improves upon the zero-forcing receiver since zero forcing can cause noise amplification if the minimum singular value of H is too small.

The condition number is the ratio between the largest and smallest singular values in the H matrix. Linear receivers (including zero forcing receivers and MMSE Receivers) are sensitive to large condition numbers. In order to account for this, we can account for the variance term. This is known as the regularization term. Our goal with the MMSE receiver is to minimize the MMSE term given by the formula in Eqn. 11 assuming the variance is greater than 0.

$$\hat{\mathbf{x}} = \text{argmin}_{\mathbf{x}} \, \|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2 + \lambda \|\mathbf{x}\|^2$$

The receiver implementation is given as a matrix product in Eqn. 13 where lambda is the variance while nothing is transmitting on both channels.

$$\hat{x} = \left(\boldsymbol{H}^H + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{H}^H \boldsymbol{y} \tag{13}$$

3. Results

Using the zero forcing receiver, we had an error rate of .45% for information sent from transmitter 1 and a 1.46% error rate for information sent from transmitter 2.

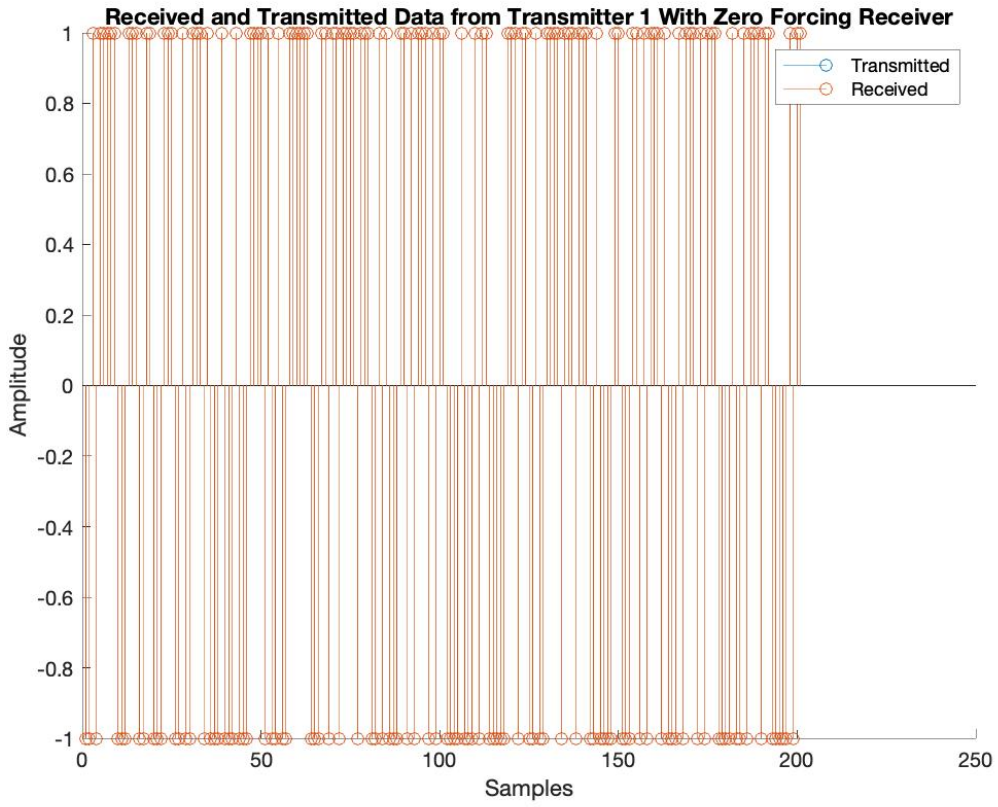

Figure 2: Received and transmitted data from transmitter 1 using a zero forcing receiver

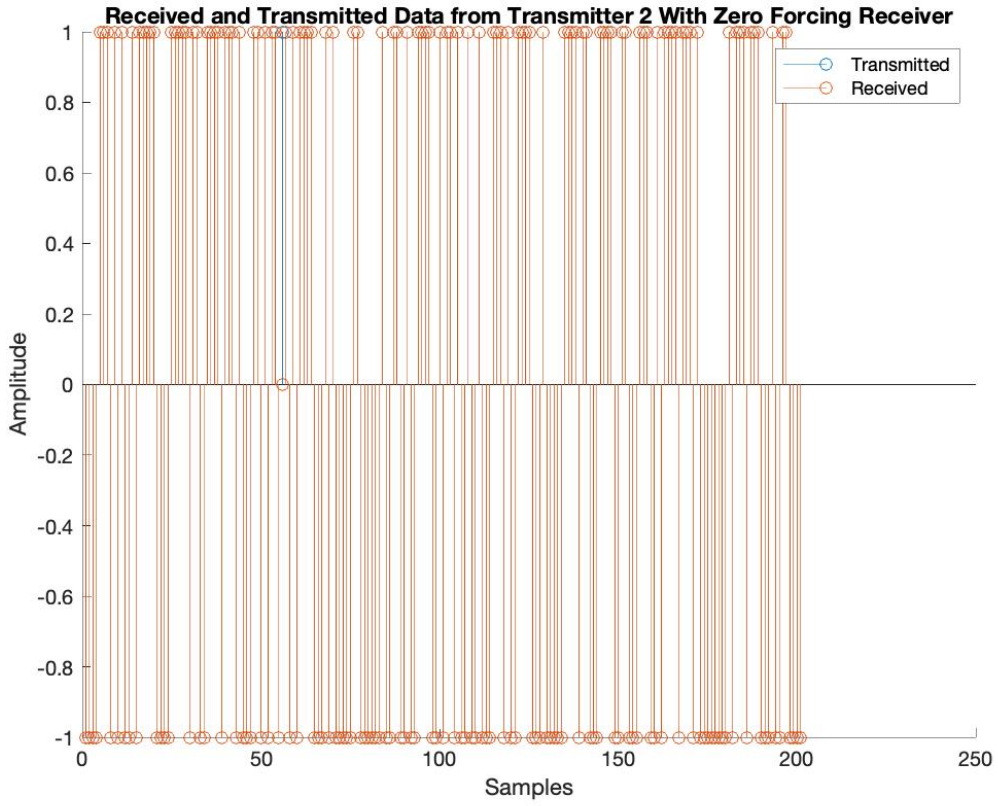

Figure 3: Received and transmitted data from transmitter 2 using a zero forcing receiver

From the MMSE detector, we had an error rate of .39% for information sent from transmitter 1 and a 1.57% error rate for information sent from transmitter 2.

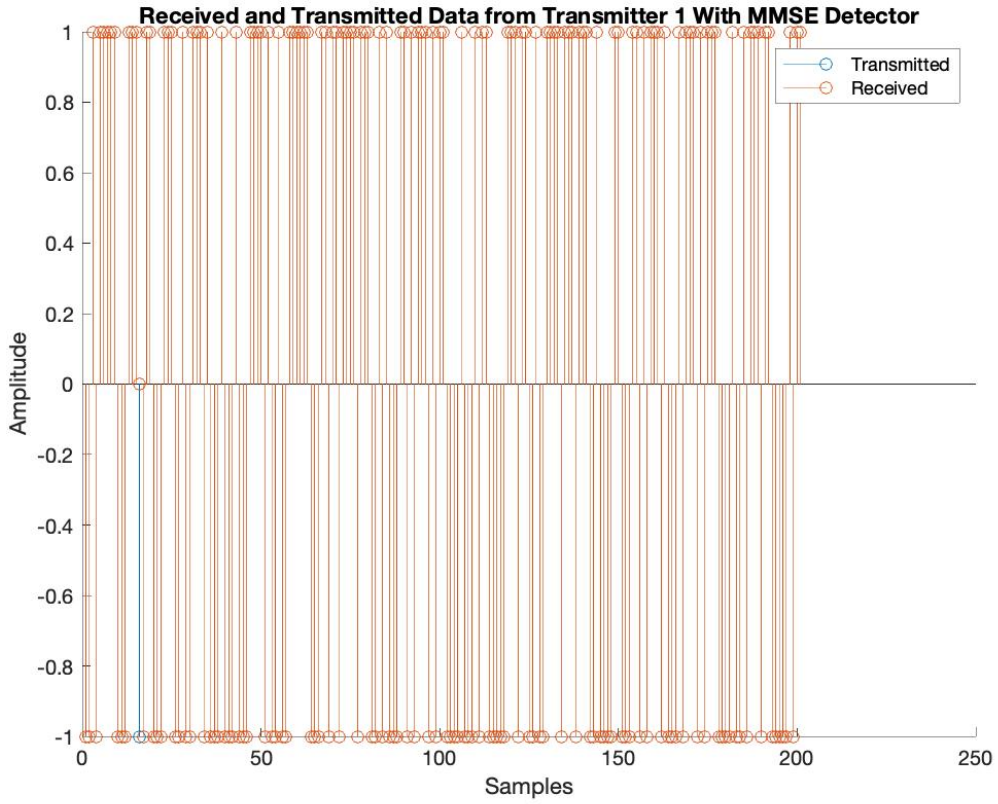

Figure 4: Received and transmitted data from transmitter 1 using an MMSE detector

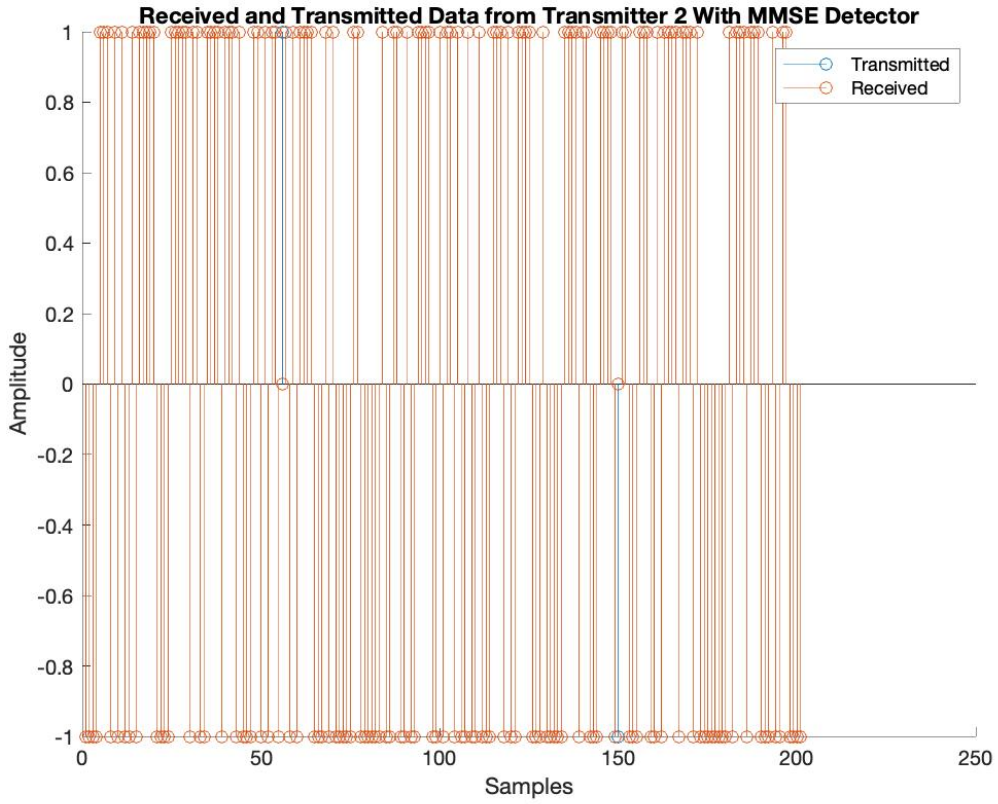

Figure 5: Received and transmitted data from transmitter 2 using an MMSE detector

4. Next Steps

To improve accuracy, we could take the windowed average of a subsection of each pulse width to make sure that the bit we sampled was not an error. The windowed average would make sure that the surrounding bits of the bit we sample is in agreement.

5. Code

Overall code for the zero forcing receiver:

```
%% Zero Forcing Receiver
clear all
load('2User2AntennaBS.mat');
%% Make vectors.
[y1, y2] = take_real(y1, y2);

%% Account for any delay.
[d1, lags1, cor1] = find_delay(y1, x1);
```

```
[d2, lags2, cor2] = find_delay(y2, x2);

% delay the signals
y1 = y1(d1:end);
y2 = y2(d2:end);

%% Creating matrices.
x = [x1'; x2'];
y = [y1'; y2'];
%% Estimate the channel response.
H = estimate_channel_response(x1, x2, y1, y2, pulseWidth);

%% Calculate the weight vectors.
% we know that H* x w = [1; 0] for x1
w1 = H' \ [1; 0];

% we know that H* x w = [0; 1] for x2
w2 = H' \ [0; 1];

%% Apply the weight vectors.
x1_hat = w1' * y;
x2_hat = w2' * y;

%% Downsample the data.
starting = ceil(pulseWidth/2) + 1;
x1_hat_round = sign(round(x1_hat));
x1_hat_down = x1_hat_round(starting:pulseWidth:end);

x2_hat_round = sign(round(x2_hat));
x2_hat_down = x2_hat_round(starting:pulseWidth:end);

%%
x1 = x1(starting:pulseWidth:end);
x2 = x2(starting:pulseWidth:end);
%% Calculate the error.
error_tx1 = calculate_error(x1_hat_down, x1);
error_tx2 = calculate_error(x2_hat_down, x2);

figure
hold on
stem(x1(1000:1200));
stem(x1_hat_down(1000:1200));
title('Received and Transmitted Data from Transmitter 1 With Zero Forcing Receiver')
xlabel('Samples')
ylabel('Amplitude')
legend('Transmitted', 'Received')

figure
hold on
stem(x2(1000:1200));
stem(x2_hat_down(1000:1200));
title('Received and Transmitted Data from Transmitter 2 With Zero Forcing Receiver')
xlabel('Samples')
ylabel('Amplitude')
```

```
legend('Transmitted', 'Received')
hold off
```

Overall code for the MMSE detector:

```
%% MMSE Detector
clear all
load('2User2AntennaBS.mat');
%% Make vectors.
[y1, y2] = take_real(y1, y2);

%% Account for any delay.
[d1, lags1, cor1] = find_delay(y1, x1);
[d2, lags2, cor2] = find_delay(y2, x2);

% delay the signals
y1 = y1(d1:end);
y2 = y2(d2:end);

%% Creating matrices.
x = [x1'; x2'];
y = [y1'; y2'];
%% Estimate the channel response.
H = estimate_channel_response(x1, x2, y1, y2, pulseWidth);

%% Calculate the weight vectors using MMSE.
lambda = var(y1(1:5000));
w = H' / (H * H' + lambda*eye(2,2));
w1 = w(1,:)';
w2 = w(2,:)';

%% Apply the weight vectors.
x_hat = w * y;
x1_hat = x_hat(1, :);
x2_hat = x_hat(2, :);

%% Downsample the data.
starting = ceil(pulseWidth/2) + 1;
x1_hat_round = sign(round(x1_hat));
x1_hat_down = x1_hat_round(starting:pulseWidth:end);

x2_hat_round = sign(round(x2_hat));
x2_hat_down = x2_hat_round(starting:pulseWidth:end);

%%
x1 = x1(starting:pulseWidth:end);
x2 = x2(starting:pulseWidth:end);
%% Calculate the error.
error_tx1 = calculate_error(x1_hat_down, x1);
```

```
error_tx2 = calculate_error(x2_hat_down, x2);

figure
hold on
stem(x1(1000:1200));
stem(x1_hat_down(1000:1200));
title('Received and Transmitted Data from Transmitter 1 With MMSE Detector')
xlabel('Samples')
ylabel('Amplitude')
legend('Transmitted', 'Received')

figure
hold on
stem(x2(1000:1200));
stem(x2_hat_down(1000:1200));
title('Received and Transmitted Data from Transmitter 2 With MMSE Detector')
xlabel('Samples')
ylabel('Amplitude')
legend('Transmitted', 'Received')
hold off
```

Taking the real components from the received signal:

```
function [y1, y2] = take_real(y1, y2)
    y1 = real(y1);
    y2 = real(y2);
end
```

Finding the delay between two signals:

```
function [delay, lags, cor] = find_delay(y, x)
    [cor, lags] = xcorr(y, x);
    [~,index]=min(cor);
    delay = lags(index);
end
```

Estimating the channel response:

```
function H = estimate_channel_response(x1, x2, y1, y2, pulseWidth)
% Note: We are assuming noise is negligible.

% transmitter 1 transmits
starting_tx1 = 5001;
message_length = pulseWidth * 128;
ending_tx1 = starting_tx1 + message_length - 1;

x1_1 = x1(starting_tx1 : ending_tx1);
y1_1 = y1(starting_tx1 : ending_tx1);
y2_1 = y2(starting_tx1 : ending_tx1);

% transmitter 2 transmits
starting_tx2 = starting_tx1 + 5000 + message_length;
ending_tx2 = starting_tx2 + message_length - 1;

x2_2 = x2(starting_tx2 : ending_tx2);
y1_2 = y1(starting_tx2 : ending_tx2);
y2_2 = y2(starting_tx2 : ending_tx2);

% calculating the channel response
% when y1 is listening to x1 speak
h11 = mean(y1_1 ./ x1_1);
% when y2 is listening to x1 speak
h21 = mean(y2_1 ./ x1_1);

% when y1 is listening to x2 speak
h12 = mean(y1_2 ./ x2_2);
% when y2 is listening to x2 speak
h22 = mean(y2_2 ./ x2_2);

H = [h11 h12; h21 h22];
end
```

Calculating the error:

```
function error = calculate_error(data_hat, data)
    error = sum(abs(reshape(data' - data_hat, 1, []))) / (length(data));
end
```

5. Resources

[0] Code at: https://github.com/naviatolin/PoW-Labs/tree/master/Lab_2A

[1] Yong Soo (2010) Channel Estimation in MIMO-OFDM Wireless Communications with MATLAB

[2] Stanford EE359 Staff (2017) MIMO Detection Algorithms