Laboratory #1

Simulation of a Simple Digital Communication System

SYSC 4600 Digital Communications

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

© Ian Marsland

Purpose and Objectives

The purpose of this laboratory experiment is to provide familiarity with some of the key building blocks of digital transmission systems.

In this lab you will write a MATLAB simulator for a simple digital communication system. Simulation is an invaluable tool for assessing the performance of communication systems in a controlled environment while avoiding expensive field trials. Furthermore, because much of the functionality of modern digital transceivers is implemented in software using digital signal processing, writing a simulator for a transmitter or receiver is very similar to actually building one.

It is important that you thoroughly understand what you are doing, as you will be building on this simulator over the remainder of the term. Take your time experimenting with your simulator. Don't rush!

Background

A simplified block diagram of a bandpass communication system is shown in Figure 1.

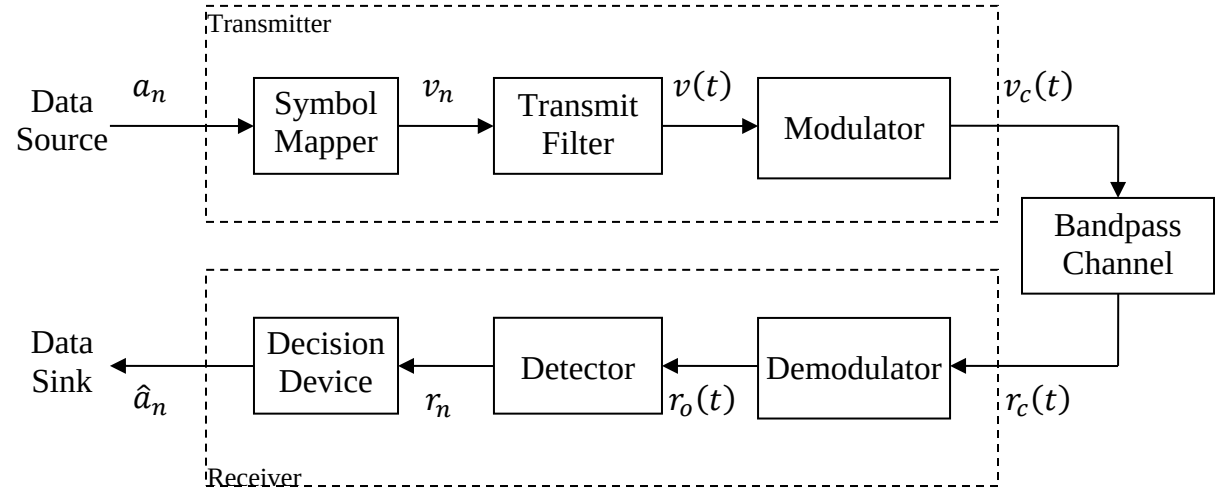

Figure 1: Simplified block diagram of a digital communication system.

Messages (sequences of bits) are to be transmitted from the data source to the data sink. The transmitter consists of a symbol mapper, which converts the bits to amplitudes, a transmit filter which generates an analogue baseband signal, and a modulator which up-converts the baseband signal to a higher frequency suitable for transmission over the channel. The receiver consists of a demodulator to down-convert the received bandpass signal back to the baseband, a detector which filters and samples the baseband signal, and a decision device which estimates the value of each transmitted bit. Each of these components is described in more detail below.

Data Source:

The transmitted message sequence consists of N_a message bits. It is given by

$$\mathbf{a} = \left[a_0\ a_1\ a_2\ \cdots\ a_{N_a-1}\right],$$

where $a_n \in \{0,1\}$ is the n^{th} message bit.

Symbol Mapper:

The *symbol mapper* converts the message bits into real-valued amplitudes. The mapping between bits and amplitudes depends on the communication scheme that is used. For this lab we will use what is known as *antipodal signaling*, where the mapping is defined by

$$v_n = SM[a_n] = \begin{cases} +1 & \text{if } a_n = 0 \\ -1 & \text{if } a_n = 1 \end{cases}.$$

That is, message bit values of 0 are converted to +1, and bit values of 1 are converted to -1.

Transmit Filter:

The *transmit filter* (also known as the *pulse shaping filter*) uses the amplitudes from the symbol mapper, $\mathbf{v} = \left[v_0\ v_1\ v_2\ \cdots\ v_{N_a-1}\right]$, to generate an analogue pulse train,

$$v(t) = \sum_{n=0}^{N_a-1} v_n h_T(t - nT)\ ,$$

where *T* is the *pulse duration* (also known as the *symbol duration* or *symbol period*), and $h_T(t)$ is the *pulse shape*. There are many different choices for the pulse shape, each with their own advantages and disadvantages. For this lab we will use a simple rectangular pulse, as shown in Figure 2. Note that the pulse shape is normalized to have unit energy.

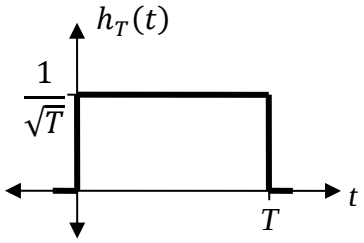

Figure 2: Rectangular pulse used by the transmit filter.

Figure 3 illustrates an example of the relationship between a message sequence, **a**, the corresponding amplitudes, **v**, and the baseband signal, $v(t)$.

Modulator:

The modulator up-converts the baseband signal to a higher frequency band by modulating the amplitude of a carrier wave. The resulting bandpass signal is given by

$$v_c(t) = v(t)\sqrt{2}\cos(2\pi f_c t)$$

where f_c is the carrier frequency.

Channel:

The bandpass signal, $v_c(t)$, is transmitted over the physical communication link (the channel). In practice, the signal is distorted as it propagates over the channel, so the signal observed at the receiver, $r_c(t)$, is different from the transmitted signal. For this lab, however, we will only consider an ideal channel, where $r_c(t) = v_c(t)$.

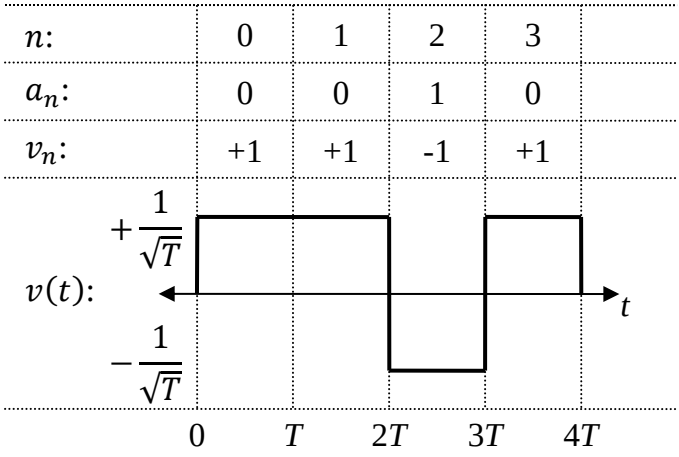

Figure 3: Example output of the transmit filter, for $\underline{a} = [0\ 0\ 1\ 0]$.

Demodulator:

The demodulator in the receiver down-converts the received bandpass signal back to the baseband. The received baseband signal is given by

$$r_o(t) = r_c(t)\sqrt{2}\cos(2\pi f_c t)$$

where f_c is the same carrier frequency as used at the transmitter.

Detector:

The detector filters and samples the received signal. The receive filter, with impulse response $h_R(t)$, acts as a lowpass filter to remove the high-frequency signal components that arise from the down-conversion process (the filter also suppresses unwanted noise that may be introduced in a non-ideal channel). The filtered signal is

$$r(t) = \int_{-\infty}^{\infty} r_o(t-\tau)h_R(\tau)\,d\tau\,.$$

In this lab we will use what is known as a *matched filter*. A filter with impulse response $h_R(t)$ is said to be matched to $h_T(t)$ if $h_R(t) = h_T(T-t)$, where *T* is the pulse duration.

The filtered received signal is then sampled, with one sample collected every *T* seconds. The first sample is collected at time $t = T$. For $n = 0,1,2,\ldots,N_a - 1$, the n^{th} sample is given by

$$r_n = r([n+1]T)\,.$$

Decision Device:

The decision device tries to determine the value of each message bit, based on the value of each received sample. The receiver output is the estimate of the transmitted message bits, $\hat{\mathbf{a}} = [\hat{a}_0\ \hat{a}_1\ \hat{a}_2\ \cdots \hat{a}_{N_a-1}]$. The decision rule for this system is:

$$\hat{a}_n = \begin{cases} 0 & \text{if } r_n \geq 0 \\ 1 & \text{if } r_n < 0 \end{cases}.$$

If everything goes well, then $\hat{a}_n$ should be equal to a_n for every *n*, indicating error-free communication.

Laboratory

In this lab you will write a MATLAB simulator for this simple digital communication system. Good computer programming practice involves developing the simulator in stages, thoroughly testing each stage before implementing the next stage. You will follow this approach when implementing your simulator, adding stages from left to right in Figure 1.

Step 1: Ideal Digital Channel

To get started, we will first simulate the ideal digital communication system shown in Figure 4.

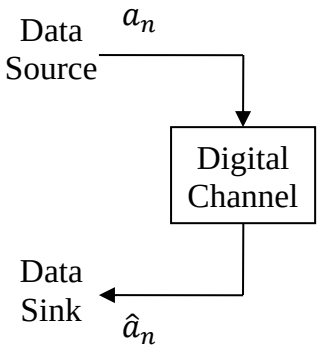

Figure 4. Block diagram of an ideal digital communication system.

The data source generates a message consisting of N_a randomly selected bits, which are transmitted over the digital channel. For this lab we will consider an ideal digital channel, so the received bits are identical to the transmitted bits. The data sink displays the received bits, and prints the number of message bits that were received incorrectly. Since the channel is ideal, this number should be zero. If it isn't, then you have a bug in your simulator, which you should correct before moving on to the next step. Finding the number of errors will be even more useful in future labs, where you will be expected to estimate the probability of error for non-ideal channels.

Enter the MATLAB code given below into a MATLAB .m file and execute it.

```
%
% Data Source
%
Na = 10;                          % Message word length (in bits)
a = randi([0 1], 1, Na);
disp(sprintf('Transmitted Message: %s', sprintf('%d ', a)));

%
% Ideal Digital Channel
%
ah = a;

%
% Data Sink
%
disp(sprintf('Received Message:    %s', sprintf('%d ', ah)));
nErrs = sum(xor(a, ah));
disp(sprintf('Number of errors: %d', nErrs));
```

Verify that there were no errors.

Question 1. Explain how the statement

```
nErrs = sum(xor(a, ah));
```

determines the number of errors. Try manually setting `ah` to something different than `a` (but the same length) before running this statement.

To simplify debugging, for the next few steps of this lab just use the message $\mathbf{a} = [1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1]$ instead of the randomly generated message. Modify your data source accordingly.

Step 2: Ideal Discrete-time Channel

Replace the 'Digital Channel' from Step 1 with the combination of a symbol mapper, an ideal discrete-time channel, and a decision device, as shown in Figure 5. The ideal discrete-time channel just passes the transmitted amplitudes without distortion, so $r_n = v_n$.

You must write code to implement the symbol mapper and decision device as described in the Background section of this lab manual. You should not change the data source or data sink. Keep $N_a = 10$ and $\mathbf{a} = [1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1]$.

Verify that there are no transmission errors before proceeding to Step 3.

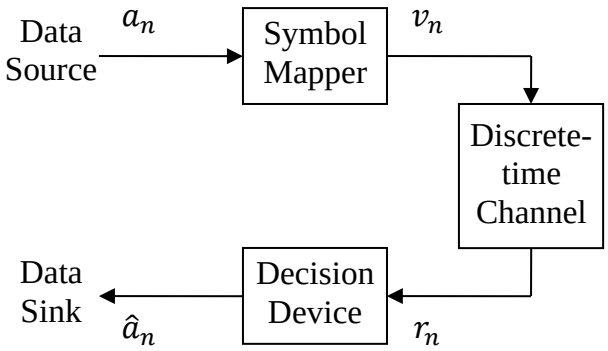

Figure 5. Block diagram of a discrete-time communication system.

Step 3: Ideal Baseband Channel

Replace the 'Discrete-time Channel' from Step 2 with the combination of a transmit filter, an ideal baseband channel, and a detector, as shown in Figure 5. The ideal baseband channel conveys with transmitted baseband signal without distortion, so $r_o(t) = v(t)$.

Add the transmit filter and detector to your simulator. Use $T = 1$ msec for the symbol duration of the normalized rectangular pulse shown in Figure 2. It is a good idea to always use normalized pulse shapes, so that the transmit filter neither amplifies nor attenuates the signal. We want the transmitted energy per symbol to be the same regardless of the pulse shape, to make it easier to study the effects of the pulse shape on system performance without the effects being obscured by potentially different transmitted signal strengths (average energy per bit).

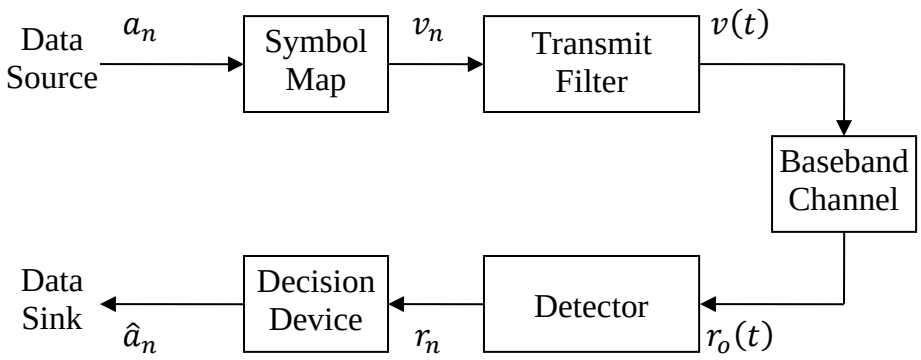

Figure 6. Block diagram of a baseband communication system.

Because it is impossible to process analogue signals in a digital computer, in MATLAB we must work with a sampled version of the signal. The baseband signal $v(t)$ will be sampled with a sampling period of $T_s = T/\eta$ seconds per sample, or $\eta = T/T_s$ samples per symbol (bit), where η is the *oversampling ratio*. We will use $\eta = 64$ for this lab. The samples are given by

$$\tilde{v}_m = v(mT_s)$$

However, because we can't actually generate $v(t)$, we can't actually sample it. Instead, we must generate the samples directly. The most useful (although not necessarily the easiest or most efficient) way of doing this is to first generate a sampled impulse train based on the amplitudes **v**, and generate a sampled version of $h_T(t)$, and then perform a discrete-time convolution of the two. This is illustrated in Figure 7, for $\mathbf{v} = [1 \quad -1 \quad 1]$ with $N_a = 3$ and $\eta = 4$.

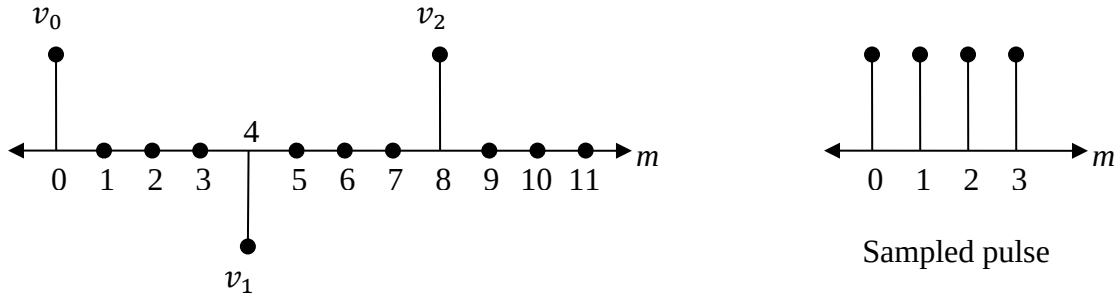

Figure 7. Examples of the sampled impulse train and sampled pulse used to generate samples of $v(t)$, for $\mathbf{v} = [1 \quad -1 \quad 1]$ and $\eta = 4$.

To generate the sampled impulse train you can use

```
upsample(v, eta)
```

where `eta` = η. The sampled pulse is just the value $1/\sqrt{T}$ repeated η times (see the MATLAB `ones()` function). Use the MATLAB `conv()` function to perform the discrete-time convolution. Note that this technique generates a signal that is $\eta - 1$ samples too long (the last $\eta - 1$ samples are all equal to 0). You should discard these samples using

```
vt = vt(1:Ns);                    % Truncate to first Ns samples
```

where `vt` is the MATLAB vector containing the samples of $v(t)$ and $N_s = N_a\eta$ is the total number of samples of $v(t)$.

Question 2. Plot $v(t)$ vs. t (in seconds). Make sure your time scale is correct (hint: how long should your signal last, and does this match what is shown in you graph?). Use the MATLAB `ylim()` command to display the y-axis over the interval $[-1.1,1.1]/\sqrt{T}$.

To implement the detector, you will have to generate a sampled version of $h_R(t)$, and use `conv()` to perform a discrete-time convolution of this with your received signal, giving an oversampled version of $r(t)$. To make the discrete-time convolution performed in the detector better mimic the continuous-time convolution that would be performed by an analogue filter, you should multiply the result of the discrete-time convolution by T_s.

Question 3. Plot $r(t)$ vs. t (in seconds). You should limit the y-axis to [-1.1,1.1]. From your graph, what are the values of $r(t)$ at the first four sampling instants, $t = [n+1]T$ for $n = 0,1,2,3$?

Down-sample $r(t)$ by a factor of η, making sure you keep only the samples at the correct sampling instants.

Question 4. Compare your first four values of r_n with the values from Question 3. Are they identical? Hint: If they aren't then you're probably sampling at the wrong times.

Step 4: Ideal Bandpass Channel

Add the modulator and demodulator to finish off your simulation of the system in Figure 1. Use a carrier frequency of $f_c = 2$ kHz. Use an ideal bandpass channel, which does not introduce any distortion, so that

$$r_c(t) = v_c(t)\,.$$

Question 5. Plot $v_c(t)$ vs. t (in seconds).

Question 6. Plot $r_o(t)$ vs. t (in seconds).

Step 5: Spectrum of the Baseband Signal

For the remainder of this lab we will use the following parameters: $N_a = 128$ bits, $T = 0.01$ seconds, $\eta = 64$ samples per symbol, and $f_c = 400$ Hz. **Use a randomly-generated message.**

We are interested in the PSD of the transmitted baseband signal, $v(t)$. One method to estimate the spectrum of a signal is to square the magnitude of the discrete Fourier transform of the sampled signal. This can be accomplished with the following MATLAB commands:

```
Vf = fftshift(fft(vt));
PSD = abs(Vf).^2 * Ts / Ns .* sinc((-Ns/2:Ns/2-1)/Ns).^2;
```

where `vt` is the MATLAB vector containing the samples of $v(t)$. The MATLAB vector `PSD` will contain an estimate of the spectrum over the interval $\left[\frac{-1}{2T_s}, \frac{1}{2T_s}\right)$, at N_s points with an increment of $\frac{1}{N_s T_s}$.

Question 7. Plot the spectrum of $v(t)$ (in dB) vs. frequency (in Hz). Make sure your frequency scale is correct. Limit the x-axis to the range from -750 to 750 Hz, and limit the y-

axis to the range from -50 dB to 10 dB. On the same graph, but in a different colour, plot the theoretical PSD,

$$\Phi_v(f) = \frac{1}{T}|H_T(f)|^2$$

evaluated at the same frequencies, where $H_T(f)$ is the Fourier transform of $h_T(t)$. Your graph should look similar (but not identical) to the one shown in Figure 8.

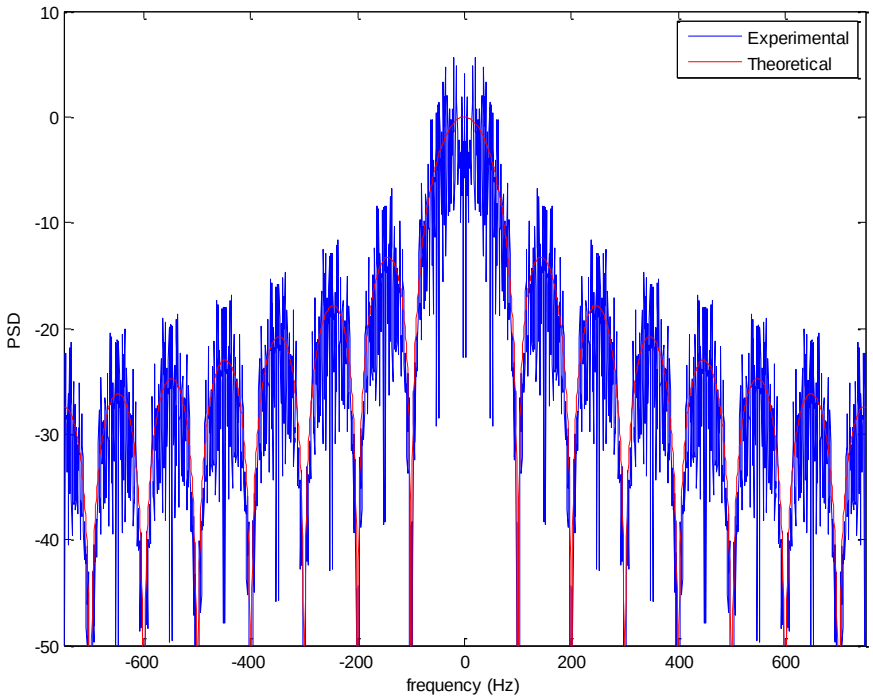

Figure 8. PSD of baseband signal.

By comparing the theoretical and experimental results, it should be apparent that although the two curves are similar, the experimental results are much more jagged. It is desirable to get a smoother curve (and hence higher accuracy) for the experimental results. Increasing η doesn't help (higher values of η does however allow us to estimate the spectrum at higher frequencies), and increasing N_a doesn't help either (increasing N_a does allow us to estimate the spectrum with a higher resolution, but this doesn't make the graph smoother). Instead we will use what is known as *Bartlett's method*, which involves taking the average of several different estimates of the spectrum.

Question 8. Rerun your simulator and generate another graph of the spectrum. It should be similar to, but slightly different than, the graph you produced for Question 7. Why is it different?

Modify your simulator by putting nearly everything in a big loop. Instead of just transmitting one message of N_a bits, you will transmit several different randomly selected messages, one after the other. Let N_f be the total number of simulated messages that are transmitted. For each message your program should calculate (but not plot) the spectrum of $v(t)$ for that message. After the loop

your program should plot the average of these spectra. Your program should also print the total number of bit errors that occurred (this should be equal to zero).

Question 9. Plot the spectrum of $v(t)$ averaged over $N_f = 10$ messages. Show the theoretical spectrum in the same graph. Are the experimental results less jagged than in Question 1? Repeat with $N_f = 100$ and $N_f = 1000$. Do the experimental results match the theoretical ones? How and why do they differ?