Laboratory #4

Synchronization

SYSC 4600 Digital Communications

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

Purpose and Objectives

The purpose of this laboratory experiment is to investigate carrier and clock recovery to achieve synchronization in digital communication systems. It builds on Laboratory #2 and #3, which you should review. You are encouraged to refer to previous lab manuals while working on this lab.

Background

For IQ modulation schemes (e.g., PSK, PAM, QAM), the transmitted bandpass signal can be expressed mathematically as

$$v_c(t) = \Re\{v(t)\sqrt{2}e^{j2\pi f_c t}\}, \quad (1)$$

where $v(t)$ is the complex lowpass equivalent baseband signal, f_c is the carrier frequency, and $\Re\{z\}$ denotes the real part of z. The complex lowpass equivalent signal can be expressed using a pulse train as

$$v(t) = \sum_{n=0}^{N_v-1} v_n h_T(t - nT) \ , \quad (2)$$

where v_n is the n^{th} transmitted symbol, N_v is the number of transmitted symbols, T is the symbol period and $h_T(t)$ is the pulse shape, which has a duration of LT seconds. The transmitted symbols are points in the signal constellation for the modulation scheme that is being used. These are represented as complex numbers.

When the signal is transmitted it travels at the speed of light which, although fast, is not infinite. As a result, there is a slight delay, known as the *propagation delay*, between when the signal is transmitted and when it is received. The signal is also attenuated, due to cable loss in a wired system or signal dispersion in a wireless one. A more realistic model than the one used in the previous labs has the received signal expressed as

$$r_c(t) = \alpha_c v_c(t - \tau_c) + w_c(t) \ , \quad (3)$$

where α_c is the signal attenuation and τ_c is the propagation delay, both of which depend on the distance between the transmitter and the receiver. They are generally unknown and can change over time (although they usually change slowly compared to the time it takes to transmit a short message).

The propagation delay causes two different impairments at the receiver. The complex lowpass signal is delayed, so it is difficult to know the exact times to sample the output of the matched filter, and a phase mismatch occurs between the carrier wave used at the transmitter and the local carrier reference signal used for demodulation at the receiver. This is illustrated by the following analysis.

The demodulated signal at the receiver is given by

$$\begin{aligned} r_o(t) &= r_c(t)\sqrt{2}e^{-j2\pi f_c t} \\ &= [\alpha_c v_c(t - \tau_c) + w_c(t)]\sqrt{2}e^{-j2\pi f_c t} \\ &= \alpha_c v_c(t - \tau_c)\sqrt{2}e^{-j2\pi f_c t} + w_c(t)\sqrt{2}e^{-j2\pi f_c t} \\ &= \alpha_c v_c(t - \tau_c)\sqrt{2}e^{-j2\pi f_c t} + w_o(t) \ , \end{aligned} \quad (4)$$

where $w_o(t) = w_c(t)\sqrt{2}e^{-j2\pi f_c t}$ is the demodulated noise, which is an modelled as complex additive white Gaussian noise (the real and imaginary parts are independent stationary Gaussian random processes with zero-mean and a flat double-sided noise power spectral density of $N_0/2$). Substituting Equation (1) for $v_c(t)$ gives in Eq. (4)

$$r_o(t) = \Re\{\alpha_c v(t-\tau_c)\sqrt{2}e^{j2\pi f_c(t-\tau_c)}\}\sqrt{2}e^{-j2\pi f_c t} + w_o(t) \ . \tag{5}$$

Since $\Re\{z\} = \frac{1}{2}[z+z^*]$, where the superscript * denotes complex conjugate, we can write this as

$$r_o(t) = \frac{1}{2}\left[\alpha_c v(t-\tau_c)\sqrt{2}e^{j2\pi f_c(t-\tau_c)} + \alpha_c v^*(t-\tau_c)\sqrt{2}e^{-j2\pi f_c(t-\tau_c)}\right]\sqrt{2}e^{-j2\pi f_c t} + w_o(t) \ , \tag{6}$$

which simplifies to

$$r_o(t) = \alpha_c v(t-\tau_c)e^{-j2\pi f_c\tau_c} + \alpha_c v^*(t-\tau_c)e^{-j2\pi(2f_c)t}e^{j2\pi f_c\tau_c} + w_o(t) \ . \tag{7}$$

The first term in Eq. (7) contains the message data (the part we're most interested in), the second term is centered around a high frequency ($2f_c$) and will be filtered out by the matched filter so we can ignore it, and the third term is the additive noise. Substituting Eq. (2) for $v(t)$ in Eq. (7) and dropping the high-frequency term gives

$$r_o(t) = \sum_{n=0}^{N_v-1} \alpha_c v_n h_T(t-\tau_c-nT)\, e^{-j2\pi f_c\tau_c} + w_o(t) \ . \tag{8}$$

The output of the matched filter, which has an impulse response of $h_R(t) = h_T(LT-t)$, is given by

$$r(t) = r_o(t) \circledast h_R(t) = \int_{-\infty}^{\infty} r_o(t-\tau)\, h_R(\tau)\, d\tau \tag{9}$$

where $\circledast$ denotes convolution. Substituting Eq. (8) into Eq. (9) gives

$$r(t) = \int_{-\infty}^{\infty}\left[\sum_{n=0}^{N_v-1} \alpha_c v_n h_T(t-\tau-\tau_c-nT)e^{-j2\pi f_c\tau_c} + w_o(t-\tau)\right] h_R(\tau)\, d\tau \ . \tag{10}$$

Rearranging the order of integration and summation gives

$$r(t) = \sum_{n=0}^{N_v-1} \alpha_c v_n e^{-j2\pi f_c\tau_c}\int_{-\infty}^{\infty} h_T(t-\tau-\tau_c-nT)h_R(\tau)\, d\tau + \int_{-\infty}^{\infty} w_o(t-\tau)h_R(\tau)\, d\tau \ . \tag{11}$$

By defining

$$h_{TR}(t) = h_T(t) \circledast h_R(t) = \int_{-\infty}^{\infty} h_T(t-\tau)\, h_R(\tau)\, d\tau \tag{12}$$

as the combined impulse response of the transmit and received filters, and

$$w(t) = \int_{-\infty}^{\infty} w_o(t-\tau)h_R(\tau)\, d\tau \tag{13}$$

as the filtered additive noise, we can write $r(t)$ as

$$r(t) = \sum_{n=0}^{N_v-1} \alpha_c v_n e^{-j2\pi f_c \tau_c} h_{TR}(t - \tau_c - nT) + w(t) \ . \tag{14}$$

To prevent intersymbol interference (ISI), the pulse shape, $h_T(t)$, is chosen so that

$$h_{TR}(nT + LT) = \delta_n = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases} \ , \tag{15}$$

where δ_n is the Kronecker delta and LT is the duration of the pulse shape, when a matched filter, with impulse response $h_R(t) = h_T(LT - t)$, is used at the receiver.

To recover the transmitted symbols (v_n) from $r(t)$ the receiver needs to know τ_c so it knows when to start sampling $r(t)$. Suppose for now that τ_c is perfectly known. Then the receiver can sample at times $t = nT + LT + \tau_c$. Sampling at these times gives the received samples

$$\begin{aligned} r_n &= r(nT + LT + \tau_c) \\ &= \sum_{m=0}^{N_v-1} \alpha_c v_m e^{-j2\pi f_c \tau_c} h_{TR}(nT + LT + \tau_c - \tau_c - mT) + w(nT + LT + \tau_c) \ . \end{aligned} \tag{16}$$

By defining $w_n = w(nT + LT + \tau_c)$ as the sampled noise, and $\phi_c = -2\pi f_c \tau_c$ as the *phase error*, we can write Eq. (16) as

$$r_n = \sum_{m=0}^{N_v-1} \alpha_c e^{j\phi_c} v_m h_{TR}([n - m]T + LT) + w_n \ . \tag{17}$$

Substituting Eq. (15) into Eq. (17) gives

$$\begin{aligned} r_n &= \sum_{m=0}^{N_v-1} \alpha_c e^{j\phi_c} v_m \delta_{n-m} + w_n \\ &= \alpha_c e^{j\phi_c} v_n + w_n \ , \end{aligned} \tag{18}$$

Multiplying v_n by $e^{j\phi_c}$ causes a rotation of the symbols in the signal constellation, and multiplying v_n by α_c causes a scaling of the symbols. If the receiver passes r_n directly to the decision device and ignores these effects, then reliable communication will be impossible. However, if the receiver knows the attenuation, α_c, and phase error, ϕ_c, their effects can be removed by first calculating

$$z_n = \frac{r_n}{\alpha_c e^{j\phi_c}} = v_n + \frac{w_n}{\alpha_c e^{j\phi_c}} \tag{19}$$

and passing z_n to the decision device instead of r_n.

From the preceding discussion, we can see that the receiver needs to know τ_c, α_c, and ϕ_c to operate correctly.

Techniques for compensating for the phase error, ϕ_c, are known as *carrier synchronization* or *carrier recovery* techniques. Traditionally, carrier synchronization is implemented using a *phase locked loop* (PLL), which is an analogue circuit that observes the received bandpass signal, $r_c(t)$, and generates a local carrier reference signal that has the same frequency and phase as the carrier in the received signal. By using this locally generated carrier reference signal instead of $\sqrt{2}e^{-j2\pi f_c t}$ when demodulating the received signal, the phase error is automatically cancelled, so

there is no need to explicitly estimate ϕ_c and cancel it from the received samples. However, it can take a long time for the PLL to initially lock onto the received carrier wave, and it can sometimes lose the lock and must resynchronize. More recently there is a movement to perform carrier recovery digitally, based on observing the received samples, r_n. We will implement one possible digital carrier recovery technique in this lab.

The receiver also needs to know the correct times to sample the filtered received signal, $r(t)$, to give the received samples, r_n. Determining the correct sampling times is known as *clock synchronization* or *clock recovery*. Like carrier recovery, clock recovery can be done with either analogue circuitry or digital signal processing. In this lab we will perform digital clock recovery.

Typically digital carrier and clock recovery must be done jointly. That is, we can't estimate α_c and ϕ_c without knowing τ_c, and we can't estimate τ_c without knowing α_c and ϕ_c. However, to understand the techniques involved, we'll start by looking at the two estimation problems separately, and then consider how to combine them.

Laboratory

Part I: Carrier Recovery

Theory:

To start, we'll assume there is only a phase error and attenuation without a delay or noise. In this case the demodulated signal can be written as

$$r_o(t) = \alpha_c e^{j\phi_c} v(t) \ , \tag{20}$$

where the high-frequency component and the noise have been ignored. The noiseless received samples from the matched filter are

$$r_n = \alpha_c e^{j\phi_c} v_n \ . \tag{21}$$

Suppose that N_v symbols, $[v_0, v_1, \dots, v_{N_v-1}]$, have been transmitted, and we have received the corresponding samples $[r_0, r_1, \dots, r_{N_v-1}]$. We want to determine $[v_0, v_1, \dots, v_{N_v-1}]$ given $[r_0, r_1, \dots, r_{N_v-1}]$. Unfortunately, we have $N_v + 2$ unknowns (the N_v symbols along with α_c and ϕ_c) but only N_v equations, so we can't directly solve the problem.

One way to avoid the problem of having an underdetermined set of equations is to use *training*. Let's suppose the transmitter and receiver agree that v_0 will always be equal to 1, and the remaining $N_v - 1$ transmitted symbols will contain the message data. Now we have N_v equations and N_v unknowns, so the problem can be solved. In particular, if we calculate

$$q = \frac{r_0}{v_0} \tag{22}$$

at the receiver (which we can do since we know v_0 =1) then q will be equal to $\alpha_c e^{j\phi_c}$ (assuming there isn't any noise). We can then divide each r_n by q, which is equivalent to dividing r_n by $\alpha_c e^{j\phi_c}$, to remove the effects of the attenuation and phase error from the remaining received samples.

Unfortunately, in practice there is also additive noise effecting the received samples, so q is an inexact estimate of $\alpha_c e^{j\phi_c}$. If the noise in r_0 is large then q will be nearly useless for compensating for the phase error. To alleviate this problem it is a good idea to use a *training sequence* of N_T

symbols instead of just one training symbol. That is, the transmitter and receiver agree on what the first N_T symbols must be, and the remaining $N_v - N_T$ transmitted symbols will contain the actual message data. The attenuation and phase offset can then be estimated by averaging over the first N_T received samples as

$$q = \frac{1}{N_T} \sum_{n=0}^{N_T-1} \frac{r_n}{v_n} \tag{23}$$

To increase the accuracy with which q estimates $\alpha_c e^{j\phi_c}$ it is desirable to make the training sequence length, N_T, as large as possible. However, the larger the length, the more time and energy is expended transmitting know symbols instead of the desired message data, so it is desirable to make N_T as small as possible. It is important to find an acceptable balance between these competing objectives.

The actual values of the transmitted training symbols, $[v_0, v_1, \ldots, v_{N_T-1}]$, are also important. It's usually a bad idea to use the same value for all the symbols (i.e., don't use $v_n = 1$ for all the training symbols). There have been numerous studies to determine the best training sequence under various system constraints and conditions, but for the purposes of this lab we'll just use a random sequence of $+1$ and -1.

Experiments:

Step 1: Before you begin, make sure you have completed Lab #2. You will use the simulator you wrote for 4-QAM. Make sure it's working properly and that your simulator is giving results that closely match the theoretical probability of error when $N_a = 1000$ and $N_f = 1000$.

Note: If you were unable to complete Lab #2 then you can use the program start.m that is provided on cuLearn as your starting simulator.

Adjust your simulator parameters to send only one message word ($N_f = 1$) that contains $N_a = 20$ bits (i.e., ten 4-QAM message symbols), and disable the additive noise.

Use $T = 0.01$ seconds, $\eta = 64$ samples per symbol, $f_c = 800$ Hz, and rectangular pulse shape with a duration of T seconds.

Make sure your simulator gives no bit errors when there is no noise.

Step 2: Generate a random training sequence of $N_T = 20$ random BPSK symbols (i.e., each symbol is ± 1). You could use the following MATLAB code to do this:

```
v_train = 1 - 2*randi([0, 1], 1, Nt);
```

Note: You should only create one training sequence, before the main loops in your program. You will use the same training sequence for all transmitted message words. Remember that the training sequence is something that the transmitter and receiver have agreed upon in advance, so it makes no sense to change it with every message.

Create a transmitted symbol sequence consisting of the $N_T = 20$ BPSK training symbols followed by the ten 4-QAM message symbols, so the transmitted symbol sequence is $N_v = 30$ symbols long. Transmit these symbols by passing them through the pulse shaping filter and modulator, then sending them over an ideal channel (no noise, no phase error, no delay). At

the receiver, demodulate the signal, pass it through the matched filter, and sample the filtered signal.

At this point you should have 30 samples. The first 20 should be the same as the training sequence and the last 10 should be the same as the message symbols. If not, you're probably starting to collect your samples at the wrong time. That is, the first sample of $r(t)$ should be taken at time $t = LT$, which corresponds to the $L\eta^{\text{th}}$ element of your MATLAB array that contains $r(t)$. Discard the first 20 received samples (corresponding to the training sequence) and pass the remaining 10 on to the decision device. Make sure there are no bit errors at the output of the decision device. Don't move on to the next step until you get this working.

Step 3: Introduce a phase error of $\phi_c = 3\pi/4$ radians. This can be accomplished by multiplying the demodulated signal, $r_o(t)$, by $e^{j\phi_c}$. Run your simulator and you should see a lot of bit errors (somewhere around 75% of the bits should be incorrect). This is expected, because a phase error that isn't compensated for will prevent reliable communication.

Step 4: Compensate for the phase error by dividing the received samples by $e^{j\phi_c}$. That is, send $z_n = r_n/e^{j\phi_c}$ to the decision device instead of r_n. Run your simulator again, and there shouldn't be any more errors.

Step 5: Instead of assuming the receiver somehow magically knows ϕ_c, use the first $N_T = 20$ received samples, along with the training sequence, to estimate q using the technique described in Eq. (23) above. Use this value of q to compensate for the phase error by sending $z_n = r_n/q$ to the decision device for the remaining received samples. Again, if you're doing everything right, there should not be any bit errors at the output of the decision device.

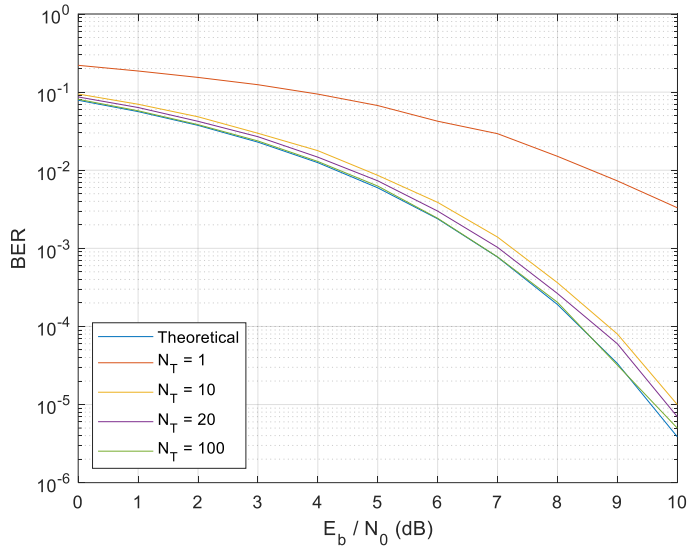
Step 6: Instead of using a fixed phase error of $\phi_c = 3\pi/4$, randomly select the phase error over the interval $[0,2\pi)$. In MATLAB, you could use

```
phi_c = 2*pi*rand(1,1);
```

Rerunning your simulator should again yield no errors.

Step 7: Test the simulator in the presence of noise. Increase the message word length to $N_a = 1000$ bits and simulate $N_f = 1000$ message words. For E_b/N_0 in the range of 0 dB to 10 dB with a step size of 1 dB, plot the BER vs. E_b/N_0 in a graph.

Repeat the experiment for different training sequence lengths ($N_T = 1$, $N_T = 10$, $N_T = 20$, and $N_T = 100$), and plot all the results in one graph, along with the theoretical BER of 4-QAM. Your results should look like the graph below. Note that the results improve, approaching the theoretical results in the absence of phase error, as N_T gets larger.



Part II: Clock Recovery

Theory:

From the introductory discussion in this lab manual, we know the receiver should sample the output of the matched filter every T seconds to get estimates of the transmitted symbols. The question is when to start collecting the samples. Theoretically the first sample should be taken at time $t = LT + \tau_c$, and each additional sample taken T seconds later. But since the receiver doesn't know τ_c this isn't very helpful.

Digital clock recovery techniques normally involve having the receiver over-sample the matched filter output, collecting multiple samples every symbol period with a high resolution, and saving these samples in memory. Digital signal processing techniques can then be used to sift through the received data, looking for the training sequence, and then deciding which samples are the appropriate ones to pass to the decision device. Since your simulator already oversamples the matched filter output (to allow you to represent the analogue signal at the output of the matched filter in a digital computer), you can just use those oversamples directly, which are sampled with a sampling period of one sample every $T_s = T/\eta$ seconds.

To illustrate one possible algorithm, let $\tilde{r}_n$ be the n^{th} oversample of the matched filter output, $r(t)$. That is,

$$\tilde{r}_n = r(nT/\eta)$$

In your MATLAB code, the first element of the array that you use to store $r(t)$ will be $\tilde{r}_1 = r(T/\eta)$. If there isn't any delay or noise then $\tilde{r}_{L\eta}$ will be the first training symbol, $\tilde{r}_{(L+1)\eta}$ will be the second training symbol, and so on. If the propagation delay is one sample (i.e., $\tau_c = T/\eta$) then the training symbols will be in $\tilde{r}_{L\eta+1}$, $\tilde{r}_{(L+1)\eta+1}$, and so on. In general, if the propagation delay is k samples (i.e., $\tau_c = kT/\eta$) then the training symbols will be in $\tilde{r}_{L\eta+k}$, $\tilde{r}_{(L+1)\eta+k}$, and so on. So, an algorithm to find the propagation delay is:

1) Set $k = 0$.
2) If the N_T samples $\left[\tilde{r}_{L\eta+k}, \tilde{r}_{(L+1)\eta+k}, \ldots, \tilde{r}_{(L+N_T-1)\eta+k}\right]$ are equal to the training symbols then

a. Stop. The propagation delay is estimated as $\tau_c = kT/\eta$.

3) Increase *k* by one and repeat from Step 2.

Unfortunately, when there is noise in the communication system, it is very unlikely that the received samples will exactly match the transmitted training symbols. As a result, instead of looking for an exact match with the training symbols, the algorithm should stop if the samples are "close" to the training sequence. A convenient measure for "closeness" in this case is the correlation between the samples $\left[\tilde{r}_{L\eta+k}, \tilde{r}_{(L+1)\eta+k}, \ldots, \tilde{r}_{(L+N_T-1)\eta+k}\right]$ and the training symbols $\left[v_0, v_1, \ldots, v_{N_T-1}\right]$, which is given by

$$\mu_k = \left| \frac{1}{N_T} \sum_{n=0}^{N_T-1} \tilde{r}_{(L+n)\eta+k} v_n^* \right|$$

If the correlation is "large" then the received samples are "close" to the training symbols. So, the receiver just needs to keep increasing *k* until it finds a correlation that is large enough. However, there are challenges in defining what is "large enough", so practical modern receivers use additional techniques to determine the propagation delay. For the purposes of this lab exercise, we'll cheat a little bit and assume there is a maximum limit on the propagation delay (that is, there is some maximum value *k*, which we'll call $k_{\max}$). For this lab you just need to calculate μ_k for all $k \in \{0,1, \ldots, k_{\max}\}$, and then find the value of *k* with the maximum value of μ_k.

Experiments:

Step 1: Save a backup of your solution for Part I. You'll need it so you can easily start again when things go wrong implementing these steps.

Step 2: Remove the random phase offset that you previously added in the demodulator (see Step 3 and Step 6 from Part I) – you won't be needing this again.

You should also temporarily disable the phase error estimation and compensation. That is, send r_n instead of $z_n = r_n q^*$ to the decision device.

Temporarily adjust your simulator parameters to send only one message word ($N_f = 1$) that contains $N_a = 20$ bits (i.e., ten 4-QAM message symbols), and disable the additive noise.

Keep $T = 0.01$ seconds, $\eta = 64$ samples per symbol, and $f_c = 800$ Hz. Use a training sequence of length $N_T = 20$.

Make sure your simulator still works (no bit errors).

Step 3: Modify your channel model so that there is a delay of exactly $\tau_c = 112\ T/\eta$ seconds. This can be done by making the received bandpass signal equal to 112 zeros followed by the transmitted bandpass signal:

```
rct = [zeros(1, 112) vct zeros(1, 500)];
```

where `vct` is the MATLAB array containing the transmitted bandpass signal, $v_c(t)$, and `rct` contains the received bandpass signal, $r_c(t)$. Note that the instruction given above also adds a bunch of zeros at the end, after the transmitted symbols. This is more realistic, since we don't usually know when the transmission ends either, so it'll prevent you later from accidentally "cheating" while trying to determine τ_c. More importantly, it'll prevent your program from

crashing if your delay detection algorithm is slightly off (which happens occasionally and is perfectly normal).

Now modify the sampler in your receiver so that it starts sampling $r(t)$ later, starting at time $t = LT + \tau_c$. If everything is working then the first 20 samples should be them same as the training sequence and the next 10 should be the same as the message symbols. Again, don't move on to the next step until you get this working. Discard the first 20 samples, and pass the next 10 samples to the decision device and make sure there are no bit errors.

Step 4: Implement the propagation delay algorithm described above. Use $k_{\max} = 500$. Print out the value of *k* found by your algorithm. It should be 112. If not, figure out why not and fix your program. Make sure there are no bit errors.

Step 5: Change the propagation delay used in Step 3 to $\tau_c = 120\ T/\eta$. Print out the value of *k* found by your algorithm. It should be 120, and there shouldn't be any bit errors.

Now change the propagation delay to $\tau_c = 114\ T/\eta$. Your program should find a value of $k = 114$, but you should also now see some bit errors. This is because the propagation delay also introduces a phase rotation that your program isn't currently dealing with. If the propagation delay is a multiple of 8 samples (since $\eta/f_c T = 8$) then the phase rotation is a multiple of 2π so it has no effect, which is why you didn't see any errors in the previous two tests.

Re-enable the phase error estimation and compensation from Part I of the lab. That is, send $z_n = r_n/q$ instead of r_n to the decision device. Make sure you're discarding the first *k* oversamples before calculating q, to compensate for the delay. That is, you should calculate q based on $\left[\tilde{r}_{L\eta+k}, \tilde{r}_{(L+1)\eta+k}, \ldots, \tilde{r}_{(L+N_T-1)\eta+k}\right]$. Now you shouldn't get any bit errors.

Step 6: Modify your channel model to implement a random delay that is chosen over the interval $\tau_c \in [0, k_{\max} T/\eta)$. You can use

```
delay = randi([0, kmax], 1, 1);
rct = [zeros(1, delay) vct zeros(1, kmax-delay)];
```

Run your program a few times (roughly 10), printing out the random delay and your estimate of *k*. They should match each time you run the program. And there shouldn't be any bit errors.

Step 7: Test the simulator in the presence of noise. Increase the message word length to $N_a = 1000$ bits and simulate $N_f = 1000$ message words. For E_b/N_0 in the range of 0 dB to 10 dB with a step size of 1 dB, plot the BER vs. E_b/N_0 in a graph.

Repeat the experiment for different training sequence lengths ($N_T = 1$, $N_T = 10$, $N_T = 20$, and $N_T = 100$), and plot all the results in one graph, along with the theoretical BER of 4-QAM. Compare your results with those from Part I where there was only a phase error. Notice that the system performance is about the same for large N_T, but much worse when N_T is small. A longer training sequence is needed for clock recovery to work effectively.

Step 8: Rejoice, for you are finished the lab.