ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 6 Modern Digital Communications
Assignment 4 October 1, 2025 (due Oct 7, 2025)

Symbol Synchronization

In Assignment 2 we implemented the building blocks of a basic communication system that uses symbol-by-symbol on pulse train signaling, depicted in Figure 1, and saw how to evaluate its performance in MATLAB/Python.

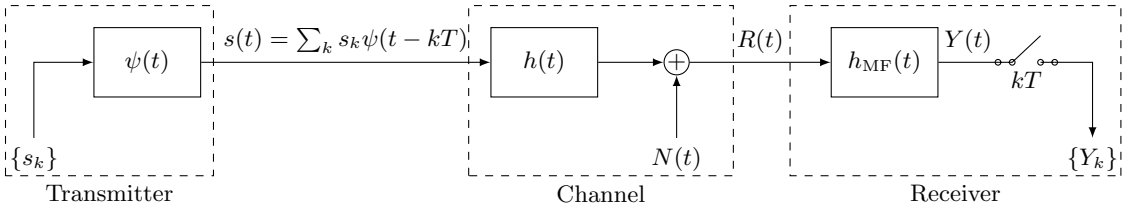


Figure 1: Communication System of Interest (ψ is a Nyquist pulse)

In our simulations we assumed that the channel only adds white Gaussian noise to the input signal and is, otherwise, perfect, i.e., $h(t) = \delta(t)$. In this case, choosing the matched filter impulse response as $h_{\mathrm{MF}}(t) = \psi^*(-t)$ guarantees that the T-spaced samples of the matched filter output are ISI-free noisy versions of data symbols, i.e.,

$$Y_k = s_k + Z_k, \qquad k \in \mathbb{Z}\,, \tag{1}$$

where Z_k are i.i.d. zero-mean Gaussian noise samples with variance $N_0/2$ in each real-valued dimension.

This assumption on the channel is not realistic. Among many other impairments (that we will see during this course), a channel always delays the transmitted signal by an amount that is typically unknown to the receiver. Typically there is also an unknown offset between the transmitter and the receiver's clock. The result is that the receiver observes a noisy and time-translated version of the transmitted signal:

$$R(t) = s(t - T_0) + N(t),$$

and the time translation T_0 is, a priori, unknown to the receiver.

Exercise 1.

1. Assume $h(t) = \delta(t - T_0)$ in the communication system of Figure 1 and ψ is a Nyquist pulse. Show that if instead of sampling the matched filter output at times $t = kT$, we sampled it at $t = kT + T_0$, the samples $Y_k = Y(kT + T_0)$ will be ISI-free (as in Equation (1)).

   Therefore, if the receiver knew the value of T_0, it could easily synchronize with the transmitter by *offsetting* the sampling time of the matched filter output. For this reason, the problem of symbol synchronization is sometimes referred to as *sampling-time offset correction*.

2. For any pulse shape $p(t)$, define its self-similarity function as

$$R_p(t) \coloneqq \int p(\alpha + t)p^*(\alpha)d\alpha.$$

   (a) Show that R_p satisfies Hermitian symmetry, i.e., $R_p(t) = R_p^*(-t)$.

   (b) Use the Cauchy–Schwarz inequality[1] to show that

$$|R_p(t)| \le R_p(0) = \|p\|^2.$$

[1] $|\langle u, v\rangle| \le ||u||\,||v||$ with equality iff u and v are collinear.

Based on what we proved in Exercise 1 part 2, one way to estimate the channel delay, T_0, at the receiver is the following: Suppose we transmit a pulse $p(t)$, known to the receiver. Then, the receiver can compute

$$\rho(\upsilon) \coloneqq \langle R(t), p(t-\upsilon)\rangle, \tag{2}$$

where $R(t)$ is the received signal. If we *only* transmitted $p(t)$ (without any data signal) the received signal would be $R(t) = p(t-T_0) + N(t)$. Accordingly,

$$\rho(\upsilon) = R_p(\upsilon - T_0) + \int N(\alpha)p^*(\alpha-\upsilon)d\alpha\ .$$

For any given value of υ, the above integral yields a zero-mean circularly-symmetric complex Gaussian random variable with variance $\|p\|^2 N_0/2$ in each dimension. Neglecting this noise term, we can conclude that $|\rho(\upsilon)|$ is maximized at $\upsilon = T_0$. Therefore, the receiver can estimate T_0 as

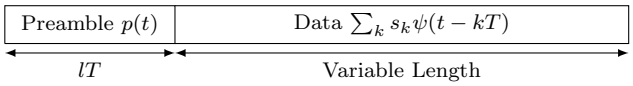
$$\hat{T_0} = \arg\max_{\upsilon} \big|\langle R(t), p(t-\upsilon)\rangle\big|. \tag{3}$$

In practice, in order for this method to work, we need to take care of a few other details.

(i) First, we neglected the channel noise. If we want to be confident that even in presence of noise the inner product $\langle R(t), p(t-\upsilon)\rangle$ is maximized when υ is essentially T_0, we have to make sure that p has a *sharp* self-similarity function. This way, at $\upsilon \neq T_0$, the self-similarity function $R_p(\upsilon - T_0)$ is much smaller than $R_p(0)$, so the noise term cannot help it exceed $R_p(0)$.

(ii) The pulse shape $p(t)$ must satisfy certain spectral properties. We are going to transmit this pulse through the channel that has a fixed bandwidth. In essence, for the same reasons that we needed a pulse-shaping filter (to make sure the spectrum of transmitted signal lies within the desired bandwidth) we have to make sure that the spectrum of $p(t)$ also has the same properties as that of $\psi(t)$.

(iii) Finally, we need to take into account the fact that $p(t)$ is not the *only* signal transmitted through the channel. It is only a *preamble* to the actual data signal. The receiver needs to use $p(t)$ to estimate the channel delay T_0 and then be able to decode the data symbols. This means that the transmitted signal is $p(t)$ followed by (a delayed) version of the data signal. The following diagram helps to better understand the structure of the transmitted signal:



For notational convenience, we can still write

$$s(t) = p(t) + \sum_k s_k\psi(t-kT), \tag{4}$$

but we implicitly assume that $s_k = 0$ for $k \leq l$ for some *training period* of l symbols. We also use the short-hand notation

$$s_{\text{data}}(t) = \sum_k s_k\psi(t-kT), \tag{5}$$

to denote the 'data' component of the transmitted signal. Consequently,

$$R(t) = p(t-T_0) + s_{\text{data}}(t-T_0) + N(t), \tag{6}$$

and, accordingly,

$$\rho(\upsilon) = \langle R(t), p(t-\upsilon)\rangle = R_p(\upsilon - T_0) + \langle s_{\text{data}}(t-T_0), p(t-\upsilon)\rangle + \langle N(t), p(t-\upsilon)\rangle \tag{7}$$

So, we also need to ensure that the contribution of the second term (due to data symbols) is negligible.

We will see how to design $p(t)$ to ensure that the above requirements are met.

Exercise 2.

Download the MATLAB/Python files for this assignment from the course webpage. Read through the script `receiver_sync_script`. This script simulates the transmission of 4-QAM data symbols using root-raised-cosine pulses through our channel of interest, evaluates the symbol-error rate, and plots the eye diagram and the constellation at the matched filter output. The line

```
received = channel(samples, SNR_dB, Es, 8*SPS);
```

simulates the channel effects. The function `channel()` simulates an AWGN channel with given SNR (`SNR_dB`) that delays the signal by a random amount. The `max_delay` parameter provides an upper-bound on the delay. So the code line above limits the channel delay to at most eight times the duration of one symbol.

If you run `receiver_sync_script`, you should see that we have a cloudy constellation at the matched filter output, and the eyes are relatively closed at time $t = 0$. Instead, the maximal vertical opening of the eye is at some other time instant. Symbol synchronization is the problem of finding this time instant, at which the eye is sufficiently open. Note that we are experimenting with a relatively high SNR (20 dB), and the relatively high symbol-error rate that you observe is almost purely due to the ISI resulting from the timing offset between the receiver and the transmitter. Indeed, setting `max_delay=0` shows that the transmission is almost error-free when there is no delay.

1. Implement `my_estimateTau.m` (for Python: `my_utilPDC.my_estimateTau`).

   You can efficiently implement these functions using the convolution/correlation operators. Carefully read the documentation for the convolution/correlation routines you decide to use, and make sure that your function correctly returns the value of υ for which $|\langle R(t), p(t-\upsilon)\rangle|$ is maximized, where $R(t)$ is the received signal (whose samples are given as the argument `rx_signal`) and $p(t)$ is an arbitrary pulse shape (whose samples are passed to the function via the argument `preamble`).

   For MATLAB, instead of convolution, one could also use the function `xcorr`.

2. One way to choose $p(t)$ is to set $p(t) = s_0\psi(t)$ for some arbitrary symbol s_0 (which may or may not come from the constellation of data symbols) and have a training period of $l = 1$ symbols. Note that this choice facilitates the implementation: the transmitter only needs to send an agreed-upon symbol s_0 at time zero (instead of data symbols). Also since $p(t)$ is the scaled version of the $\psi(t)$, it automatically satisfies all the spectral properties we need.

   (a) Show that with this choice of $p(t)$,

   $$\rho(\upsilon) = \langle R(t), p(t-\upsilon)\rangle = |s_0|^2 R_\psi(\upsilon - T_0) + \sum_{k>0} s_k s_0^* R_\psi(\upsilon - T_0 - kT) + \langle N(t), p(t-\upsilon)\rangle. \tag{8}$$

   (b) Assuming ψ is a Nyquist pulse, show that

   $$\rho(T_0) = |s_0|^2 R_\psi(0) + \int N(\alpha) s_0^* \psi^*(\alpha - T_0) d\alpha.$$

   This shows that, at least at $\upsilon = T_0$ the contribution of data symbols to the inner product is zero.

   (c) Check that $\rho(T_0 + nT)$, $\forall n > 0$, takes a similar form.

   (d) Using MATLAB/Python, plot the self-similarity function of the root-raised-cosine pulse used in our communication system.

   (e) If ψ has a sharp self-similarity function, for υ in vicinity of T_0 the contribution of the second term (summation) in (8) is still negligible. Thus, we can approximate

   $$\rho(\upsilon) \approx |s_0|^2 R_\psi(\upsilon - T_0) + \int N(\alpha) p^*(\alpha - \upsilon) d\alpha, \qquad \text{for } \upsilon \text{ in vicinity of } T_0.$$

   So, ignoring the presence of noise, we can again say that $\upsilon = T_0$ is a local maximum of $\rho(\upsilon)$. Therefore, we may hope that our estimation method still works.

   To test this, uncomment the line

```
preamble_symbols = [1];
```

in the script `receiver_sync_script`. This will add the desired preamble signal (with $s_0 = 1$) to the data signal. Is the delay estimated correctly? Using the results obtained in the previous questions, can you explain what is happening?
Hint: You may also want to plot the result of the convolution/correlation in the function `my_estimateTau`.

3. To overcome the issues observed above, we need to make sure that the inner product between the time-shifted preamble signal $p(t - \upsilon)$ and the "data" part of the signal is negligible for all values of υ.

   Consider the preamble signal constructed randomly as

$$p(t) = \sum_{i=0}^{l-1} B_i \psi(t - iT) \tag{9}$$

   where B_i are independent and identically distributed BPSK symbols, with $\Pr\{B_i = -1\} = \Pr\{B_i = +1\} = 1/2$. Note that with this choice we still guarantee that $p(t)$ satisfies the spectral properties we need.

   (a) Compute the (random) self-similarity function of p, R_p and show that

$$E\left[R_p(t)\right] = lR_\psi(t).$$

   (Note that the expectation is taken over the choice of symbols B_i.) Conclude that if ψ has a sharp self-similarity function then on average this construction results in a preamble $p(t)$ that has a sharp self-similarity function as well.

   (b) Let $q(t)$ be any random process generated independently from $p(t)$ (in particular it can be the data signal s_{data} or the noise). Show that

$$E\left[\langle q(t), p(t - \upsilon)\rangle\right] = 0.$$

   Thus, we have seen that such a randomly constructed preamble, on average, has all the properties we desire. Let us also accept, without proof, that if l is large, the behaviour of such a preamble is close to average. Therefore if we had been able to use such a truly random preamble we would have had a good estimation of channel delay. The problem is that the preamble has to be shared between the transmitter and the receiver, so it cannot really be generated randomly. In contrast, we can use what is so called pseudo-noise (P/N) sequence. A P/N sequence is generated deterministically by using Linear Feedback Shift Registers (LFSR) but 'looks' like a pure random sequence. Therefore, the transmitter and receiver can generate such sequences that look random locally, and be sure that they both get the same sequence (by initializing the LFSRs in the same state).

   We have generated[2] one such sequence of length $l = 127$ for you that is stored in the variable `pn_seq` when you run the script `receiver_sync_script`. To use a preamble generated based on this sequence, uncomment the line

```
preamble_symbols = pn_seq;
```

   in the script `receiver_sync_script`. Run the script with this preamble, and observe how well the receiver estimates the channel delay and, as a result, the eyes will be centered. If you have time, try to reduce the SNR and figure out up to what SNR the time estimator still works well.

Exercise 3. We have seen in the previous exercise a pretty robust method for estimating the channel delay. The core step of our estimator was computing the inner products between the received signal and time-shifted copies of the preamble, and determining the time shift for which the inner product is maximized. These inner products can be efficiently computed in software, but this requires the receiver to have access to samples of the received signal (i.e., $\{R(nT_s)\}$)

[2] See `help commsrc.pn` to learn how to generate such sequences yourself in MATLAB.

and necessitates implementing the matched filtering as a digital filter. In some implementations, the matched filter is still a part of the analog front-end and the digital block only receives the samples of the matched filter output, namely $\{Y(nT_s)\}$.

As computing the inner products $\langle R(t), p(t-\upsilon)\rangle$ in the analog front-end is impractical, it is natural to wonder if there is a way to estimate the channel delay using the matched filter's output samples? In this exercise we give an affirmative answer to this question.

We assume the preamble is in the form of

$$p(t) = \sum_{i=0}^{l-1} b_i \psi(t - iT).$$

We want to show that we can compute $\rho(mT_s) = \langle R(t), p(t - mT_s)\rangle$ using only the matched filter output samples $Y(nT_s)$ and the $\{b_i\}$ sequence.

1. Show that the output of the matched filter is

$$Y(t) = \langle R(\alpha), \psi(\alpha - t)\rangle.$$

2. Compute $\rho(\upsilon) = \langle R(t), p(t-\upsilon)\rangle$ and show that

$$\rho(mT_s) = \sum_k b_k^* Y(mT_s + kT).$$

This shows that we can compute (the samples of) the inner product $\langle R(t), p(t-\upsilon)\rangle$ using the samples at the *output* of the matched filter.