ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 17 Modern Digital Communications
Assignment 11 December 3, 2025 (due Dec 9, 2025)

Wireless Communications with USRP SDR Boards

In this assignment we implement an OFDM transceiver to communicate data over the wireless channel using USRP Software-Defined Radio boards.

Before getting started, download the assignment files[1] from the course webpage. We have written most of the code for you and you just need to complete it by using some of the subroutines that you have already implemented in the previous assignments, or implementing a couple of new subroutines.
As in the previous assignments, we provide the solutions for every function that you should write. You can use these solutions to test a whole working system, maybe just substituting one function at a time with your implementation. For MATLAB, the script `function_mapper_usrp` is used to select for each function whether to use the provided solution or your own implementation. For Python, the functions that you need to write are grouped under `my_utilUSRP.py`.

You will first test your transceiver using a *simulated channel* that models the subsystem from the reconstruction input at the transmitter to the sampler output at the receiver. This simulated channel takes a sequence of symbols and delivers the corresponding sequence that has been affected by filtering, additive white Gaussian noise, and carrier frequency offset. Once you have debugged your code and are sure that you have error-free transmission over the simulated channel you can (optionally) switch to transmission over the air.

For the actual transmission, we give you the routines for communicating with the boards, i.e., routines to transfer the symbols to the transmitting board and to read from the receiving board the corresponding sequence (at the symbol rate). You also have the choice between connecting two boards to the same machine or using separate computers to transmit and receive. If you are not curious to use the hardware, you can skip the last part of the assignment dealing with the USRP boards. We provide you with a very realistic channel simulator which models the imperfections of the hardware as well as the characteristics of the propagation environment.

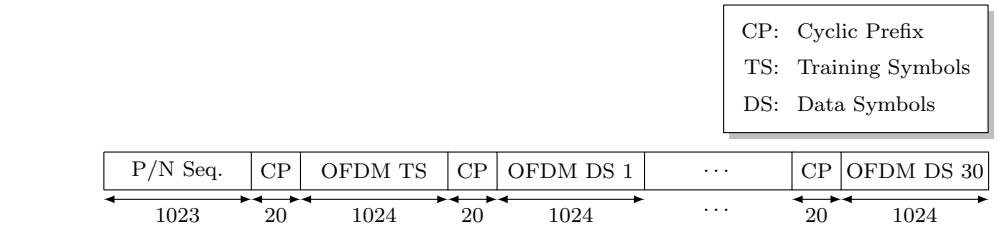
The Big Picture

The channel between two USRP boards is an ISI channel. We use an OFDM system with $N = 1024$ carriers and a cyclic prefix of length 20 to combat the ISI. The symbol duration of the system is 2 μsec, i.e., we transmit data at a rate of 500‘000 symbols per second. Therefore, each OFDM block will last $(1024 + 20) \times 2 \times 10^{-6} = 2.088 \times 10^{-3}$ seconds. For simplicity, we fix the number of transmitted OFDM blocks to 30 data blocks plus one training block used for channel estimation.

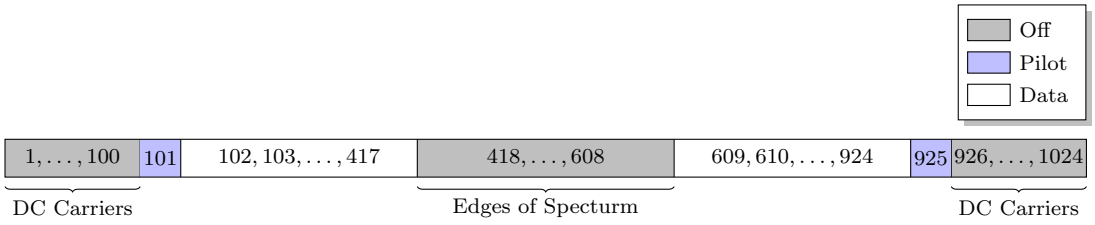
The receiver needs to know approximately when the first OFDM block starts, so that it knows which channel outputs belong to which OFDM block. (We say approximately, because small delays are accounted for by the channel impulse response.) To this end, as we have learned in one of the previous assignments, we prepend the first OFDM block with a preamble (a P/N sequence). By correlating the received signal with the P/N sequence, the receiver can detect the start of the first OFDM block within the received signal. In addition, the preamble allows the receiver to estimate the carrier-frequency offset (CFO) between the transmitting and the receiving USRP boards. All in all, the structure of the transmitted signal is as shown in Figure 1a.

As we have seen in the previous assignment, the residual CFO results in a rotation of data symbols (after the DFT), by an amount that increases linearly from one OFDM symbol to the next. To correct this rotation, we use certain carriers, called pilot carriers, to transmit symbols that are known to the receiver. In our system we use two pilot carriers at indices[2] 101 and 925. In order to control the power spectral density of the transmitted signal, one can choose to turn off certain carriers. In our case we chose not to use the carriers with indices 1 to 100 (DC), 417 to 607 (spectrum edges), and 926 to 1024 (DC). In total we use 634 out of 1024 carriers, two of which are pilot carriers, leaving 632 carriers for data (see Figure 1b).

[1] `usrp_assignment.zip`
[2] We use MATLAB-style indexing, i.e, our 1024 carriers are numbered from 1 to 1024.



(a) Structure of a Frame



(b) OFDM Symbol Structure

Figure 1: Signal Structure and Carrier Assignments

Transmitter

The function `transmitter` takes input bits and converts them to symbols. Since we have fixed the number of OFDM data blocks and the number of usable carriers per block, the maximum number of bits that can be transmitted depends only on the constellation. If we feed the transmitter with too many bits, the bits in excess are disregarded. If we feed it with fewer than the maximum number of bits, randomly generated bits are appended.

The overall structure of the transmitter's software is depicted in Figure 2a.

- We use two global structures `ofdmc` and `usrpc` to store and share the parameters of the transmitter and USRP boards among different routines. `ofdmc` stores parameters such as OFDM block length, cyclic prefix length, ..., that we have discussed here, as well as the OFDM training symbols (`ofdmc.trainingSymbols`) and the synchronization preamble (`ofdmc.preamble`). `usrpc` stores parameters such as carrier frequency, clock rate, ..., for the USRP boards. You should not need to access the parameters in `usrpc` and must be careful *not* to change them; otherwise, it might undermine the proper functioning of the boards.

  MATLAB `ofdmc` is initialized by calling `ofdmConfig()`. If you need to access its content in your functions, you can include the following lines at the beginning of your function:

  ```
  global ofdmc;
  if isempty(ofdmc)
      ofdmConfig();
  end
  ```

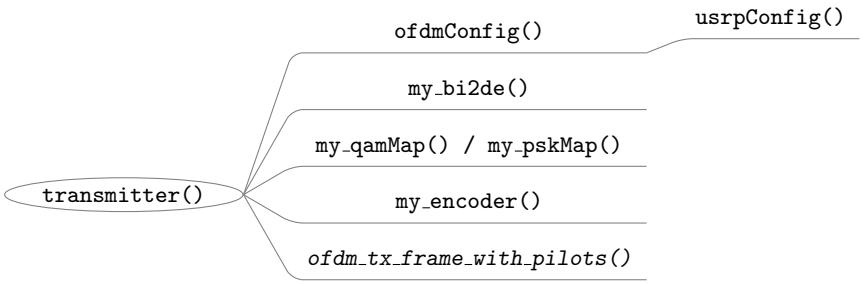
  Python Just `import ofdmc` to make the corresponding structure available.

- The transmitter first adjusts the length of the input bits (by trimming or padding with random bits), and converts them to decimal numbers between 0 and $M-1$, where M is the constellation size (`ofdmc.M`). It then maps them to data symbols from the desired constellation[3] (defined in `ofdmc.constellationType`). Up to this point, we only need to use the functions that we had implemented at the beginning of the semester (Building your own Communication Toolbox).

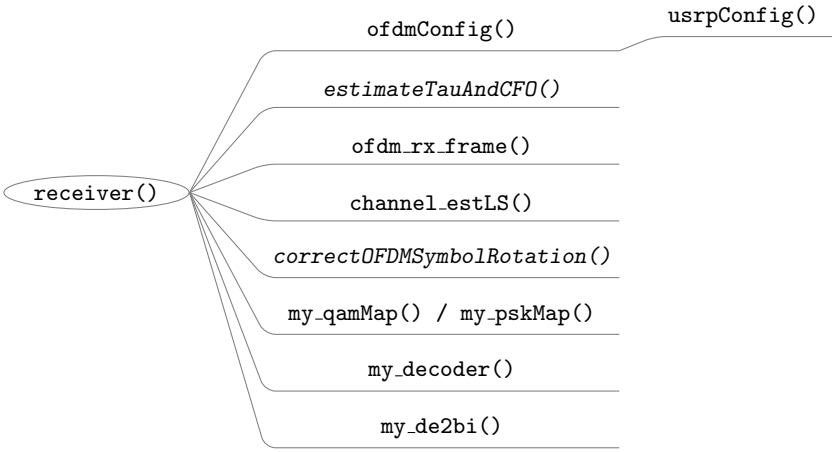
- Next, the transmitter calls the function `ofdm_tx_frame_with_pilots`. This is a modified version of the function `ofdm_tx_frame`, that you have implemented in an earlier assignment, modified so as to be consistent with the structure shown in Figure 1b.

- Finally, the OFDM blocks are appended to the synchronization preamble `ofdmc.preamble` (shown as P/N Seq. in Figure 1a).

[3]For the purposes of this assignment, we use 4-QAM, but if you wish, you can experiment with higher-order constellations and try to obtain higher bit rates.



(a) Transmitter



(b) Receiver

Figure 2: Overall Software Structure. Italicized functions are those to be written.

Receiver

The function `receiver` takes the samples of the received signal (after down-conversion to baseband) and outputs the estimated bits.

The overall structure of receiver's software is depicted in Figure 2b. Its workflow is as follows:

- The receiver finds the carrier-frequency offset (CFO) and determines the OFDM-block boundaries. Both are done with the help of the P/N sequence. The CFO is then corrected. (There is still a residual CFO that will be corrected after the DFT, see below.)
- The time-domain samples that correspond to OFDM blocks are converted to frequency domain. This is done via the function `ofdm_rx_frame` that you have already implemented in an earlier assignment.
- The coefficients of the "parallel channels" are estimated using the estimator `channel_estLS` already developed. (We cannot use `channel_estMMSE` since we do not know the delays and the path-strength variances of the multipath channel at hand.) The frequency-domain symbols are equalized by dividing each symbol by the corresponding channel-coefficient estimate.
- The residual CFO is removed. If unaccounted for, it shows up as a constellation rotation by an amount that increases linearly from one OFDM block to the next.
- Finally, the receiver extracts the data symbols (by removing the pilot symbols and the training symbols) and estimates the data symbols using a minimum distance decoder. The estimated data symbols are then mapped back to bits.

Assignments

Exercise 1. To complete the transmitter, implement the function `ofdm_tx_frame_with_pilots`.

This is basically the same function as the `ofdm_tx_frame` that you have implemented in an earlier assignment, with the difference that it inserts the pilot symbols on the specified carriers.

Exercise 2. Implement the function `estimateTauAndCFO`.

You have implemented a similar algorithm when you decoded the GPS signals. The main idea is that, if we denote by $p(t)$ the preamble (synchronization signal), the inner-product between $e^{j2\pi\epsilon f_c t}p(t-\tau)$ (a delayed and modulated version of the signal) and $e^{j2\pi\nu t}p(t-\upsilon)$ is maximized when $\nu = \epsilon f_c$ and $\upsilon = \tau$.[4] Thus if $r(t)$ is the received signal, the receiver can estimate the beginning of the sent signal and the CFO by finding

$$(\hat{\nu}, \hat{\tau}) = \arg\max_{(\nu,\tau)} \left|\left\langle r(t), e^{j2\pi\nu t}p(t-\tau)\right\rangle\right| .$$

Recall also that to solve the above maximization problem, we first evaluate the inner product for a range of ν values (the argument `cfo_range`), by using the correlation function. This will allow us to find the maximizing τ and a rough estimate of the maximizing ν. We then refine the estimation of ν by approximating the shape of the inner product (as a function of ν) with a parabola around the peak, and finding the ν that maximizes the quadratic approximation.

Exercise 3. Implement the function `correctOFDMSymbolRotation`.

As already discussed, due to a residual CFO, the frequency-domain data symbols will be rotated by an amount that increases linearly with the index of the OFDM block. Thus, we use pilot carriers to send fixed symbols used to estimate and correct this rotation.

In our setting, we have *two* pilot symbols, and your function will be called with `pilot_indices` and `pilot_symbols` of size 2. In this first implementation, you can use any of the two pilot symbols for estimating the rotation. For the chosen symbol, you simply divide the noisy channel output by the input, and find the phase of the result. It is instructive to (a) plot (or print) the (estimated) rotation of the OFDM blocks versus the block index and verify that it increases linearly from block to block and (b) look at the phase rotation for the two pilot symbols of the same OFDM block and verify that it is the same, i.e., it is carrier independent.

Exercise 4. In this exercise, we see how we can improve the phase estimator of Exercise 3 by taking advantage of having multiple pilot carriers.

Suppose we want to estimate R (a complex-valued scalar) given the observables

$$\boldsymbol{Y} = R\boldsymbol{X} + \boldsymbol{Z}$$

where $\boldsymbol{X} \in \mathbb{C}^k$ is a known vector and $\boldsymbol{Z} \in \mathbb{C}^k$ is a noise vector. We would like to make a *least-squares* (LS) estimation, i.e., find $\hat{R}_{\mathrm{LS}} \colon \mathbb{C}^k \to \mathbb{C}$ such that

$$\|\hat{R}_{\mathrm{LS}}(\boldsymbol{Y})\boldsymbol{X} - \boldsymbol{Y}\|^2$$

is minimized (for each realization of $\boldsymbol{Z}$). Show that

$$\hat{R}_{\mathrm{LS}}(\boldsymbol{Y}) = \frac{\boldsymbol{X}^\dagger \boldsymbol{Y}}{\|\boldsymbol{X}\|^2}$$

is the least-squares estimate of R given $\boldsymbol{Y}$.

In our phase-correction problem, in general, we have k pilot symbols that we can stack in the vector $\boldsymbol{X}$ and we want to estimate the phase rotation θ from the observables

$$\boldsymbol{Y} = e^{j\theta}\boldsymbol{X} + \boldsymbol{Z}$$

(where $\boldsymbol{Y}$ is the vector of received frequency-domain symbols at pilot indices). Therefore, the LS estimation of θ will be

$$\hat{\theta}_{\mathrm{LS}} = \arg\left\{\frac{\boldsymbol{X}^\dagger \boldsymbol{Y}}{\|\boldsymbol{X}\|^2}\right\} = \arg\{\boldsymbol{X}^\dagger \boldsymbol{Y}\}.$$

Modify your correction algorithm of Exercise 3 to use the LS estimation of the phase using all pilot symbols.

[4] For simplicity we use the continuous-time signals in our discussion — you know how to translate the ideas to discrete-time domain. Also, observe that here we only do a symbol-level synchronization. This is sufficient, as we only need to find the boundaries between OFDM blocks. Further delays (within the duration of one symbol) are taken care of by OFDM channel equalization.

Once you have implemented all the missing functions, test your transceiver using the script `test_sim`. This script uses a simulated channel, implemented in `channelSimulator`. It simulates the channel from the input of `usrpTx` to the output of `usrpRx`. You can control which impairments are included in the channel, by specifying the arguments `snr`, `cfo`, or `clockOffset`.

It would be useful to print out the intermediate calculation results (in particular the delay and CFO estimates in `estimateTauAndCFO`) and plot the transmitted and received signal constellation to ensure everything is working correctly. You can also change the parameters `cfo` and `clockOffset` if you want to test the system without CFO or clock offset. If your algorithms work correctly, you should have error-free transmission with a SNR value of about 20 dB.

USRP Board

After you made sure that your implementation works well on the simulated channel, you can switch to transmission via the USRP boards. In order to make it easier to compare the decoded information with the source information, we are going to transmit a text file.
There are two options concerning the use of the USRP boards[5]:

1. You use two computers, one to transmit and one to receive. In this case, you call `run_tx` on the transmitting machine and `run_rx` on the receiving machine.

2. You use a single computer to transmit and receive, with two USRP boards connected to it, one to transmit and one to receive. In this case, you call `run_single_pc`. If only one board is connected/detected, it will switch to the loopback mode, that is, the same board transmits and receives (still over the air, using its Tx/Rx antennas).

Note that if you are using two different models of USRP boards (one for Tx and the other one for Rx), you might need to increase the parameter `rangeCFO` to the values which are indicated in the file `ofdmConfig.m / ofdmc.py`. This is due to the fact that the carrier frequencies generated by the two boards might differ by more than the default values we have set there initially.

It is important to realize that the MATLAB/Python code does not run in real time. It does not matter how long it takes to produce the samples. Once produced, they are downloaded to the transmitting board which sends them out in real-time. The receiving board collects and stores the samples. Of course this has to happen in real-time. Once the collection is complete, the samples are transferred to the MATLAB/Python code which processes them off-line.

In a commercial implementation, the receiving board has to listen constantly for incoming signals, it must detect whether a signal is of interest or not, and if so it must react fast enough to capture the signal. To avoid such complications that are outside the scope of this assignment, the transmitting board keeps sending many identical copies of the frame (structure in Figure 1a) and the receiver records for a duration of about 5 seconds. This ensures that the receiving board captures multiple copies of the intended frame. The receiver routines then take out a chunk of signal of length four times that of the frame, making sure that it is not taken from the very beginning or from the very end. This way, the `receiver` routine is not given huge amounts of data. The reason for taking the data from the middle section is to avoid transients due to the switching on/off of the signals.

[5]See the course webpage (`http://moodle.epfl.ch`) for instructions on how to install the drivers to work with MATLAB/Python.