Milestone 3 (Modulation/Demodulation)

Due Date : 5th June 11:59 pm.

Version 1: Modulation-Demodulation

UPDATES

- DEMODULATION: Since we are doing quadrature demodulation, the multiplication with the complex exponential gives us I[n] + jQ[n]. Hence, after the low pass filtering, we need to take the absolute value of the output to calculate the actual demodulated samples (refer slides 24 and 25 here).

STARTER CODE

Starter Code can be downloaded here : [.zip]

GOAL

- The goal for this version of the milestone is to build the `MODULATE`, `DEMODULATE` and `DETECT_THRESHOLD` blocks (ref: Overview). You'll implement these blocks in `receiver_mil3.py` and `common_txrx_mil3.py`.
- Once you are done, your code should work just like the audiocom demo. That is, you should be able to:
  - Send monotone using:

    `python sendrecv.py -m 100 -c 1000 -s 256 -q 200`
  - Send a text/image using:

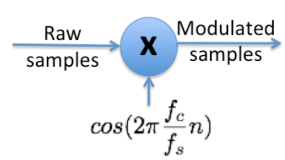
    `python sendrecv.py -f testfiles/<filename> -c 1000 -s 256 -q 200`
- Also, if you feed the transmitted samples directly to the receiver (i.e., without introducing the channel) in `sendrecv.py`, you should see no bit errors.
  - You can do this by modifying the line "`demod_samples= r.demodulate(samples_rx)`" to "`demod_samples = r.demodulate(mod_samples)`"

`common_txrx_mil3.py`

In this file, you will implement the following functions :

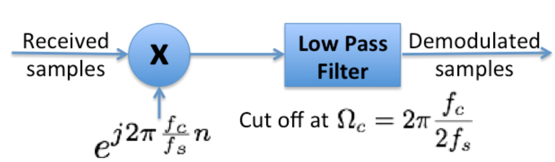
- `modulate`: modulate the samples onto a carrier signal of a given carrier frequency; the resulting modulated samples can be played on the speaker
  - Input : carrier frequency, sample rate (number of samples transmitted by the speaker per second), raw samples (before modulation)
  - Output : modulated samples for the speaker

  The modulated samples can be generated from the raw samples by multiplying with a local carrier, as shown below.



- `demodulate` : demodulate the audio signal samples with the given carrier frequency
  - Input: carrier frequency, sample rate (number of samples captured by the microphone per second), received samples from the microphone
  - Output: demodulated samples

  The demodulated samples can be obtained from the received samples using the following two-steps (ref: quadrature demodulation, here).



Low Pass Filter

In frequency domain, this low pass filter has a frequency response given by

In frequency-domain, this low pass filter has a frequency response given by

$$H(\Omega) = \begin{cases} 1 & |\Omega| \le \Omega_c \\ 0 & \text{otherwise} \end{cases}$$

You need to implement this filter in time-domain, with the cut off frequency at $\frac{1}{2}2\pi\frac{f_c}{f_s}$. The corresponding unit sample response is

$$h[n] = \begin{cases} \frac{sin(\Omega_c n)}{\pi n} & n = -L, \cdots, -1, 1, \cdots L \\ \frac{\Omega_c}{\pi} & n = 0 \end{cases}$$

This is a linear, time-invariant (though non-causal) system, so you need to convolve the input to the low pass filter with the above $h[n]$ to get the demodulated samples. In other words, if $r[n]$ are the received samples, then

$$demodsample[n] = r[n]e^{j2\pi f_c n/f_s} * h[n]$$

You may chose $L = 50$ (i.e. filter length = 101) for the project.

NOTE : Since we are doing quadrature demodulation, the multiplication with the complex exponential gives us I[n] + jQ[n]. Hence, after the low pass filtering, we need to take the absolute value of the output to calculate the actual demodulated samples (refer slides 24 and 25 here).

receiver_mil3.py

In this file, you'll implement the following module.

- `detect_threshold` : computes the centers of the two clusters (corresponding to *0*s and *1*s) in the demodulated samples, and also the threshold.
  - Input : demodulated samples
  - Output : `one` (center of the *1*s cluster), `zero` (center of the *0*s cluster), `thresh = (one+zero)/2`

  This function performs clustering as mentioned in the Primer here. You need to implement the 2-means clustering algorithm (you may search online for details of the algorithm, for e.g. here).

SUBMISSION INSTRUCTIONS

1. Login to coursework.stanford.edu using your SUNet ID.
2. Click on *Sp13-ENGR-40N-01* in the top menu to enter the ENGR 40N website on Coursework.
3. Click on *Drop Box* in the left menu to access your online drop box for this course.
4. Upload your files: `receiver_mil3.py, common_txrx_mil3.py`. Do NOT rename the files, do NOT create sub-folders (upload the two files into your main drop box folder for the course)
5. *If you are implementing anything special or will also submit Milestone 3 - Source and Channel Coding, place a `README.txt` that describes what you implemented.*

As always, if you encounter any problem, post on Piazza!