ELEC50002 Communications - Laboratory

Laboratory Handout 2 (Lab Session 3 + Lab Session 4)

Lab 2: AM Simulation and USRP

In this lab you will build AM and FM communication system simulators. In order to achieve this, you will learn how to use sub-VIs in the LabView environment.
Open **LabVIEW NXG 3.1** and create **a new project** called "**Lab2**". *Note: All the files created for solving the following exercises should be included in the Lab2 project.*

Some files/VIs that will be used in this and subsequent labs have been provided to you. You can access the files provided on MS Teams. Go to the MS Teams > ELEC50002 Communications Laboratory > Files. You need to download Lab2.zip for the tasks given in this handout.

Note: To use any of these files/VIs in any project, download the specific files/VIs from the server and save them in the same folder as the project. If you refresh the Files pane in LabView, with the project open, you will see these files/VIs appear under the project.

Reminder from Lab 1
Recalling the theory of AM, the standard equation for modulating a single tone is given by:

$$\boldsymbol{s(t) = [A_c + A_m \cos(2\pi f_m t)]\cos(2\pi f_c t)},$$

where A_m and A_c are the amplitudes, and f_m and f_c are the frequencies of the message signal and the carrier waveform, respectively.

AM modulated signal is already acquired In Exercise 3 of Lab 1, and now demodulation is needed to retrieve the original message signal.

Exercise 1: AM Demodulators

Create a demodulation module to recover the original message signal. There are two possible methods:

1. **Coherent Demodulation**: Multiply the AM signal with the carrier signal followed by a low pass filter.
2. **Envelope Detection**: Rectify the AM signal, low pass-filter it and remove the DC component to obtain the envelope.

Create two VI modules **'AMCoherent.gvi'** and '**AMEnvelopeDetection.gvi**' for the two different types of demodulation techniques.

Exercise 1a: Coherent Detection

Instructions:

1. Open the "AMCoherent.gvi" file.
2. Perform AM signal and carrier signal multiplication:
    a. Add the **Multiply** function to the diagram.
    b. Create two controls by searching for **Waveform Array Terminal** and right click on the functions and select "Change to Scalar". Name them as "AM signal" and "carrier signal", respectively. Their corresponding graphs will appear on the panel window.
3. Connect the output of the multiplication to a **Filter** module, and configure the filter as "Lowpass" "Butterworth Filter". Create controls for the "order" and low "cutoff frequency" of the filter.
4. Subtract the DC component from the main signal. The DC component can be retrieved by using the **Amplitude Measurements** module, and configuring the module as "AC DC and RMS" and "Single Shot".
5. Add waveform graph indicators to view the final results in both time and frequency domains. (When you test this module later with an input signal, you will be able to observe the message signal in both time and frequency domains after demodulation).

Important! *Pay attention to scaling the amplitude appropriately in order to recover the desired message signal.*

6. Transform your VI into a sub-VI as in Exercise 3 of Lab 1 (Inputs: AM signal, carrier signal, low cut-off frequency, order. Outputs: demodulated signal in time, demodulated signal PSD).

Tasks:

☞ Add the block diagram to your logbook.

☞ Explain briefly the mathematical theory behind this demodulation technique. Moreover, why are we using a low pass filter and why do we have to get rid of the DC component? Why do you need to scale the message amplitude?

Exercise 1b: Envelope Detection

This method consists of a **half wave rectifier** and an **RC filter.**

Instructions:
1. Open the 'AMEnvelopeDetection.gvi' file.
2. Since the multiplication with the impulse response of the rectifier introduces a loss of π in the signal, we need to amplify it before rectification by the same amount. Thus, multiply the signal with π.
3. To implement the rectifier, convert the waveform to an array using the **Waveform Properties** function, then use a **For loop** that runs through all the elements of the AM signal, and selects only the positive values (set the negative values to 0). Thus, inside the **For loop**, you will need to put a condition that checks if an element is positive (have a look at the **Case Structure** or the **Select** function under "Program Flow" in the functions palette).
4. Convert the array (of the rectified AM signal) back to a waveform using the **Build Waveform** function.
5. Build a low pass filter using the **Butterworth Filter** with the same values used in **Exercise 1a**. Also remove the DC component.
6. Transform your VI into a sub-VI (Inputs: AM signal, order, low cut-off frequency. Outputs: demodulated signal in time, demodulated signal PSD).

Tasks:

☞ Explain briefly the mathematical theory behind this demodulation technique and all the steps implemented.

☞ Add the block diagram to your logbook.

Exercise 2: AM Simulation

In this exercise, all the sub-VIs you have prepared so far will be used, and you will be able to observe the entire process of amplitude modulation and demodulation. Add the **AM** sub-VI from Lab 1 into this project.

Instructions:
1. First, create a new file and name it "**AMTopLevel.gvi**".
2. Drag **AM**, **AMCoherent** and **AMEnvelopeDetection** from the file tab on the left side of the screen, and place them on the block diagram.
3. Create necessary control and indicator terminals for each node, and make the necessary connections.
4. Create a while loop with a control that allows you to stop the loop via a stop button. Run the code and observe the process.

Tasks:

☞ Add the block diagram to your logbook.

F Set the parameters as in the following table and run the code.

Carrier amplitude	2
Message signal amplitude	Follow instruction below
Sample frequency	200k
Samples	1k
Butterworth low-pass filters	Order: 5 Cut-off frequency: 3 kHz
Carrier frequency	10 kHz
Signal message frequency	1 kHz

☞ Observe the time domain and PSD representations of the coherent and envelope demodulated signals as the message signal amplitude changes from 1 to 4 with steps of 1. As you change the message signal amplitude do you observe any changes between the two modulation techniques? If yes, explain the reason.

Exercise 3: AM Communications via USRP

In this exercise, all the required modules have already been provided to you. Your task is to run the modules using the settings provided and explain how the modules work from the code.

Some background information on the modules concerning the USRP:

niUSRP Open Tx Session initiates the USRP device, generates a session handle and an error cluster that are propagated through all four VIs. When you use this VI, you must add a control called “device name” that you will use to inform LabView about the USRP Device ID (either a name or an IP address).

niUSRP Configure Signal is used to set the parameter values. Attach four controls and three indicators to this VI as shown in Figure 1. The controls set the **IQ rate** (in Sample/s), the **carrier frequency**, the **gain**, and the **active antenna**. When the VI runs, the radio returns the actual values of these parameters (called “coerced parameters”). The actual parameters may or may not match the values put as controls. If an input value exceeds the capability of the radio, it will choose the nearest acceptable parameter value, as the coerced value, rather than returning an error.

niUSRP Write Tx Data sends the baseband signal to the USRP for transmission. Placing this VI in a while loop allows a block of baseband signal samples to be sent over and over until the “stop” button is pressed. Note that the while loop is programmed to terminate if an error is detected. Baseband signal samples can be provided to the **Write Tx Data** either as an array of complex numbers, or as a complex waveform data type. Right click on the function and select **Replace**. This will allow you to choose the data type of the function. The input data should be complex in order to introduce I and Q data to the USRP (which corresponds to the real and imaginary parts, respectively). If the baseband discrete time signal is expressed as

$$\tilde{g}[nT_x] = g_I[nT_x] + jg_Q[nT_x]$$

then the continuous time transmitted signal from the USRP is

$$g(t) = Ag_I(t)\cos(2\pi f_c t) - Ag_Q(t)\sin(2\pi f_c t)$$

In this expression, the constant A is set by the “gain” parameter (in dB) and f_c is the carrier frequency. The sampling interval T_X is the reciprocal of the “IQ rate”. Be aware that the USRP internal circuit implements some functions for transmitting and receiving the signal (Appendix A contains details of

the USRP's internal operations). The signal g(t) produced by the USRP is a continuous time signal, that is, digital to analog conversion (DAC) is done inside the USRP itself.

niUSRP Close Session terminates transmitting operation once the while loop ends. Note that the VI should be terminated using the STOP button rather than with "Abort Execution" on the toolbar, so that the Close Session VI will correctly close the data structures that the VI uses

niUSRP Open Rx Session works in the same way as **niUSRP Open Tx Session.**

niUSRP Configure Signal works in the same way as in the Tx**.**

niUSRP Initiate sends the parameter values you selected to the receiver, and starts it running.

niUSRP Fetch Rx Data retrieves the message samples received by the USRP. Placing this VI in a while loop allows message samples to be retrieved one block at a time until the "stop" button is pressed. **Number of samples** control allows you to set the number of samples retrieved at each step of the while loop. **Fetch Rx Data** can provide message samples to the user as either an array of complex numbers, or as a complex waveform data type. Right clicking and then selecting **Replace** allows you to choose the data type for the message samples. This function does not output the received signal directly. Instead, *USRP itself multiplies the received signal with the carrier, based on the configurations you set in the panel control, and applies a low pass filter to it* (see Appendix A).

niUSRP Abort stops the acquisition of data once the while loop ends.

niUSRP Close Session works in the same way as in the Tx.

Instructions:

1. Download the provided files and add them to this project. Open "**USRP_AM_Rx.gvi**" and "**USRP_AM_Tx.gvi**".
2. In the front panel of each module, set the values of the controls as in the following:

Control	**Tx**	**Rx**
Device names	NI2900	NI2900
IQ rate	500k	500k
Carrier frequency	400M	400M
Gain	0	0
Active antenna	TX1	RX2
Sample rate	500k	N/A
Samples	200k	200k
Low cutoff frequency	N/A	1400
Modulation index	1	N/A
Message frequency	1000	N/A

Tasks:

☞ Run the code and explain in your logbook how the transmitter and receiver work. **Hint:** the USRP does all the modulation and demodulation on its own so the modules only needs to provide the complex data for the USRP to modulate. At the receiver the USRP will return the demodulated data.

☞ Transmit a single-tone message signal with frequency 5kHz and modulation index 1 (you may have to adjust the low cutoff frequency of the filter). Run the receiver, and plot the demodulated signal in time, and its PSD. Adjust the time and PSD plot to the relevant region so that the plot is not just a blur, and rename the axes appropriately. Add the plots to your logbook.

☞ To observe the effect of noise in the demodulated signal, increase the receiver's gain to 20 dB (your receiver will start to detect other weaker signals in addition to the transmitted signal), and adjust the X axis of the demodulated message (in time domain) to show values between 0 second (s) and 0.004 s. Then, change the modulation index (μ) value and observe the effects on the plots of the demodulated signal in both time and frequency. From what value of μ can noise be clearly noticed in the plots? Copy the image of the noisy demodulation into your logbook.

Exercise 4: Listening to AM Music (Bonus)

Inside the lab, there is a USRP broadcasting an AM signal. The signal is a piece of music that you can listen to.

Instructions:

1. Open and set the USRP_AM_Rx_Music.gvi panel setting to the configuration below, and try to find the message signal.

***Hint**: LabView might have problems detecting the right soundcard settings automatically to play your audio. **Before running the VI**, plug-in the earphones, and try to play any audio file first (e.g. a Youtube video), preferably with LabView closed. This will ensure that the right soundcard setting is selected.*

Control	Rx
Device names	NI2900
IQ rate	100k
Carrier frequency	400M
Gain	Adjust as necessary
Active antenna	RX1
Samples	5M

Note: *LabView requires all the samples to be acquired before playing the signal, hence it may take some time before you hear the first note. Once the output signal graph starts changing, it indicates the signal is being played, and you should hear the music.*

Task:

☞ Add the graph of the demodulated signal both in time and frequency domain to your logbook and identify the frequency of the message signal and the song playing.

APPENDIX

Handling Errors in Labview

LabView allows controlling the errors that may occur for different reasons during the execution of a code. Just as data values flow through a VI, so does the error information, and this is the reason why most functions include error input and output terminals. If LabVIEW detects an error, the node passes the error to the next node. The next node does the same thing, and so on. At the end of the execution flow, LabVIEW reports the error. Thus, by wiring the error information from the beginning of the GVI to the end, and by setting appropriate indicators, it is possible to obtain details on the errors in the Panel tab.

In particular, the following functions can be useful in the following exercises:

a. When handling multiple errors, the **Merge Errors** function (available from the function palette) allows merging multiple error clusters into a single one, which can thus be propagated till the end of the function itself.
b. If you need to convert an error code to an error cluster, you will need the **Build Error Cluster** module in the function palette.

It is a good practice to properly handle errors when writing a GVI. Even if it is not a necessary condition in order to obtain the right result.