Lab 3: Coded System

Report Due: 21:00, 5/5, 2017

1 Goal

In this lab, we would like to implement coded system with convolutional code and Viterbi algorithm for the decoding. Our major goal is to compare the performance of convolutional coded systems with different rates or memory sizes.

2 Archietecture and Experiment Setup

This lab includes the encoder and the decoder, the blocks with red text in Figure 1. On the other hand, the transmitted messages in the USRP implementation are images. First, you implement assigned convolutional codes in LabVIEW simulation on your own. For USRP implementation, first you build some modules to convert given images into bits, and then deliver the data over-the-air using the convolutional coded system on USRP.

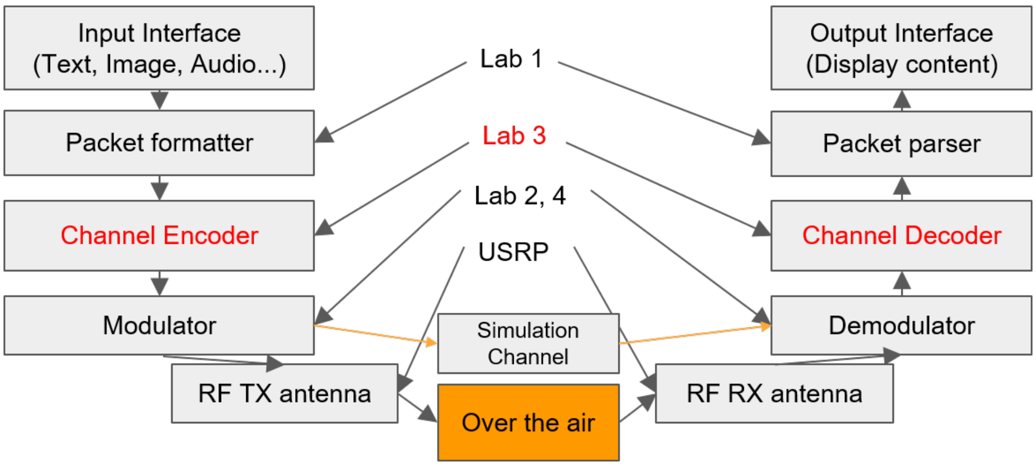


Figure 1: Block diagram

3 Implementation

In this lab, we will implement of three kinds of convolutional codes with different rates or memory sizes:

a) $[5, 7]$

b) $[13, 17]$

c) $[25, 27, 33, 37]$

After that, we will examine the performance of these codes with simulation and experiments. The rest of this section is organized as follow: First, we will show you how to program the $[5, 7]$ encoder and decoder in LabVIEW. We will also introduce a simulator for you to calculate the BER of the three kinds of convolutional codes. Second, we introduce the coded system to be implemented with USRP.

3.1 LabVIEW Simulation

3.1.1 Convolutional Encoder

Encoders of convolutional codes in LabVIEW can be implemented easily with the use of shift register and XOR modules.

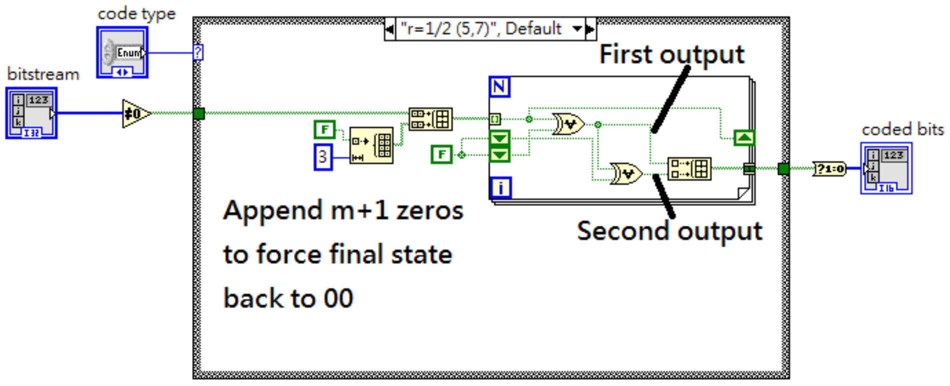


Figure 2: Encoder of [5, 7] convolutional code

As shown in Figure 2, the encoder can be built easily by a `for`-loop with 2 shift registers. The purpose of the shift registers are to serve as memories. Note that every block of input bits should be appended with $m+1$ zeors to force the final state of convolutional encoder back to S_0.

3.1.2 Convolutional Decoder

As you learn in class, an efficient algorithm to decode convolutional codes is the Viterbi algorithm. In this part, we will show you how to implement this algorithm in LabVIEW. [5, 7] decoder is explained in detail, while the rest are left to you as exercises.

Viterbi Algorithm The basic idea is to find the path with minimum cost during each stage and obtain the best result. There are two things we need to prepare before starting to run the Viterbi algorithm. The first is the state transition map and the second is the output map. The state transition and corresponding output of [5, 7] code are listed in Table 1-2. These maps help us build the Trellis diagram required to run Viterbi algorithm easily. When calculating the cost of a state transition with a given observation, the state transition map will tell us what input causes the state to change, and the output map will tell us the corresponding output bits. With the above information, we can accumulate the cost and continue to find the best path.

	S_0	S_1	S_2	S_3
S_0	0	1	-	-
S_1	-	-	0	1
S_2	0	1	-	-
S_3	-	-	0	1

Table 1: State transition map of [5, 7] code

Now we are ready to implement Viterbi algorithm. The three steps of the algorithm are:

1) Initialization

2) Update cost and record path

3) Back tracking

	S_0	S_1	S_2	S_3
S_0	0	3	-	-
S_1	-	-	2	1
S_2	3	0	-	-
S_3	-	-	1	2

Table 2: Output map of $[5, 7]$ code

The first step is to initialize two tables. The two tables record respectively the cost of best path to one stage and the state index in the previous stage where the best path is from. The second step is to complete the two tables by going through all possible states and accumulating the cost under hard or soft decision criterion. The last step is to choose the path with minimum cost from first table. Use the index and the second table to find the index from the last stage to the first stage. Equivalently, the back tracking process will result in a sequence of state transitions. We can use the state transition map to find out the message bits from the sequence of transitions. This completes the algorithm.

Recall that every block of message bits is appended with $m + 1$ zeros. As a result, the last $m + 1$ bits can be viewed as known bits. In summary, the decoding criterion is to select the most possible path on the trellis diagram that produces the output closest to observations at receiver in certain distance metric, depending on soft decision or hard decision.

Soft/Hard Decision Decoder The differences of hard decision and soft decision are in the criterion of decoding strategy. For hard decision, the received complex symbols are first demodulated to a bitstream. The cost function is the **minimum Hamming distance** of demodulated bits and output bits of a state transition. As for soft decision, the cost function is the **minimum Euclidean distance** of received complex symbol to synthetic symbol generated from the modulator with output bits of a state transition. It can be proved that soft decision decoding outperforms hard decision decoding, but this is out of the coverage of this handout. However, the following example will show you that soft decision decoding can perform better than hard decision decoding.

Suppose the message bits are 001, and we use $[5, 7]$ codes with BPSK modulation (+1 for 1, -1 for 0). At receiver side, the noisy symbol is (-1.0,0.1,-1.0,0.1,1.0,0.8). Table 3 lists all possible cases in hard or soft decision decoding. In hard decision, the minimum Hamming distance criterion will select "100" as most possible result. However, the true message bits are 001, therefore results in two bit errors. On the other hand, the soft decision decoder uses minimum Euclidean distance criterion and the most possible case is "001", which is the correct answer.

Possible bits	**Output**	**Hard decision**	**Soft decision**
000	000000	4	$1.1^2 + 1.1^2 + 1.0^2 + 0.8^2 = 4.06$
001	000011	2	$1.1^2 + 1.1^2 + 0.2^2 = 2.46$
010	001101	3	$1.1^2 + 2.0^2 + 0.9^2 + 2.0^2 + 0.2^2 = 10.06$
011	001110	3	$1.1^2 + 2.0^2 + 0.9^2 + 1.8^2 = 9.26$
100	110111	1	$2.0^2 + 0.9^2 + 0.9^2 + 0.2^2 = 5.66$
101	110100	3	$2.0^2 + 0.9^2 + 0.9^2 + 2.0^2 + 1.8^2 = 12.86$
110	111010	4	$2.0^2 + 0.9^2 + 2.0^2 + 1.1^2 + 1.8^2 = 13.26$
111	111001	4	$2.0^2 + 0.9^2 + 2.0^2 + 1.1^2 + 2.0^2 + 0.2^2 = 14.06$

Table 3: Example given message bits=001, BPSK, $[5, 7]$ convoltional codes

3.2 USRP Implementation

3.2.1 Image Source in LabVIEW

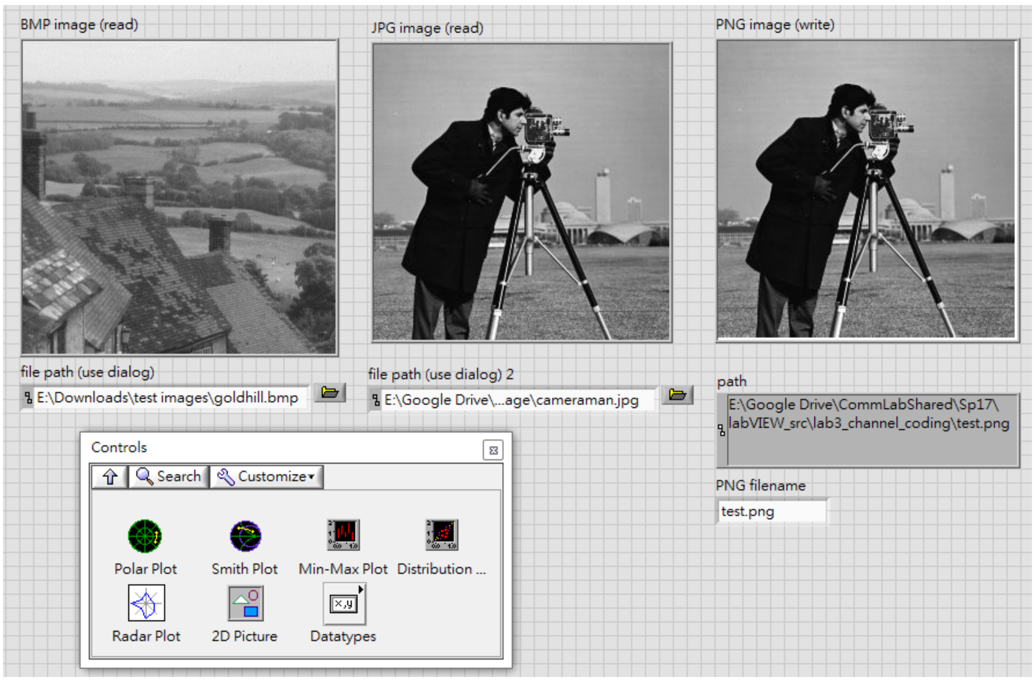
Introduction of Image Modules An image can be viewed as a two-dimensional digital data, but it is stored in the computer as a series of 0's and 1's. So how does computer know a series of bits being an image, and print it on a screen with the correct size and color? Similar to the header of a packet, an image has a header which contains crucial information including size and color. Different kinds of images have different format of the header. In LabVIEW, there are three kind of images which have built-in modules to read or write them easily:

1)JPG, 2)BMP and 3)PNG. The details of these formats are out of the coverage of this lab and we will only show you how to use built-in LabVIEW modules to read .bmp image file. The others have similar procedures and are left to you as exercises.

Image I/O Example In this part we will show you how to use LabVIEW to read and write images. Also, the example is already provided for you as reference. See `example_image.vi`.

The following describes the steps to read or write .bmp image in LabVIEW(see Figure 3 and 4):

1. **File path:** The first step is to create a control to specify the location of the file to be read.
2. **Read BMP:** Use `Read BMP File.vi` block to read .bmp file. The output of this block is a special cluster in LabVIEW called “Flatten pixmap”. “Flatten pixmap” can be drawn on control panel in LabVIEW or written to file easily with other modules. Figure 3 is an example to draw flatten pixmap.
3. **Unfaltten graph:** Flatten graph can be “unflatten” to extract information of an image such as color, mask, etc. We can use “Unflatten pixmap” to separate the header and data of an image apart.
4. **Flatten graph:** Unflatten graph can be flatten with “Flatten pixmap” block. Note that in LabVIEW, only flatten pixmap can be written to a file.
5. **Write BMP:** To write an image to file, you should fill in the header of the image and specify the path and filename correctly. So, first you should use “Flatten pixmap” to put header and data to a cluster. Then use `Write BMP file.vi` to write an image.



file path (use dialog): E:\Downloads\test images\goldhill.bmp

file path (use dialog) 2: E:\Google Drive\...age\cameraman.jpg

path: E:\Google Drive\CommLabShared\Sp17\labVIEW_src\lab3_channel_coding\test.png

PNG filename: test.png

Figure 3: Example: image modules in control panel

3.2.2 USRP **Image Transceiver**

Now we are ready to do experiment on our implementation of convolutional encoder and decoder. In this lab, we will transmit image from one antenna to the other. Also, we will combine the modules we built in previous labs such as packet formatter/parser and modulator/demodulator to build this image transceiver. Figure 5 is the control panel of this system. There is a tab to select the type of convolutional codes to be used. You can also decide how much pixels should a packet carried. The other parameters are similar to those in Lab1 and Lab2. For indicators, there are two blocks to show the original image and the received image. Below them is a bar to show you the progress of transmission. For convenience, this system only support three kinds of image:

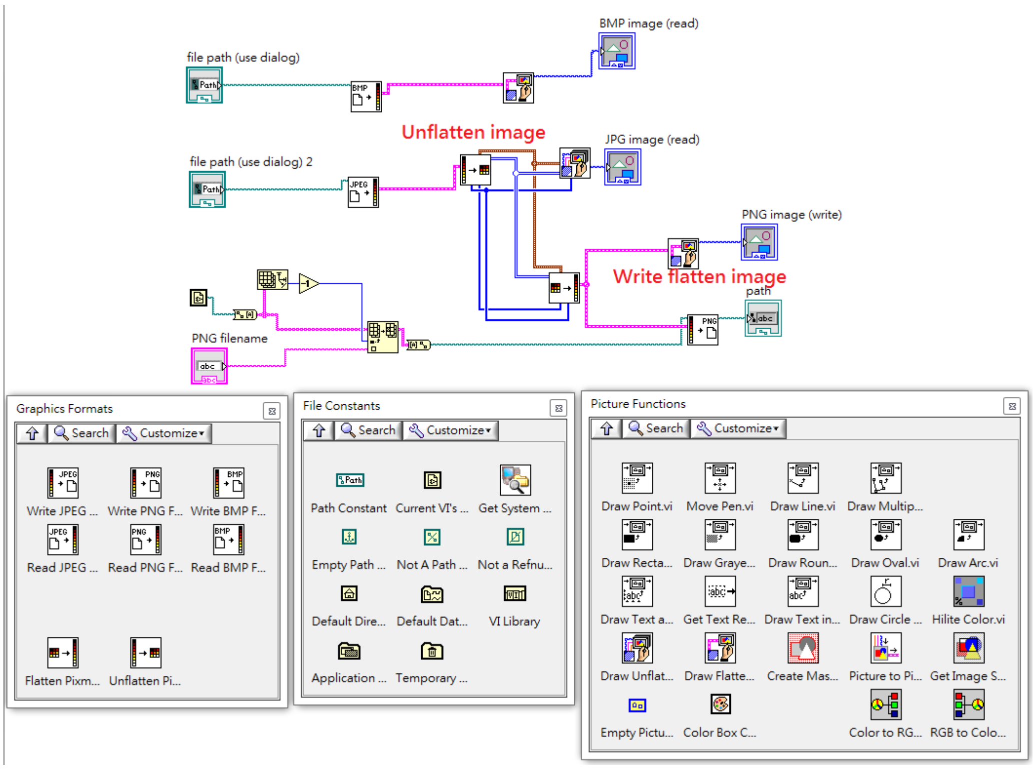


Figure 4: Example: image modules in block diagram

1)JPG, 2)PNG and 3)BMP. You are free to modify the system to do more experiment of your convolutional code. We recommend that simulation should be performed in advance.

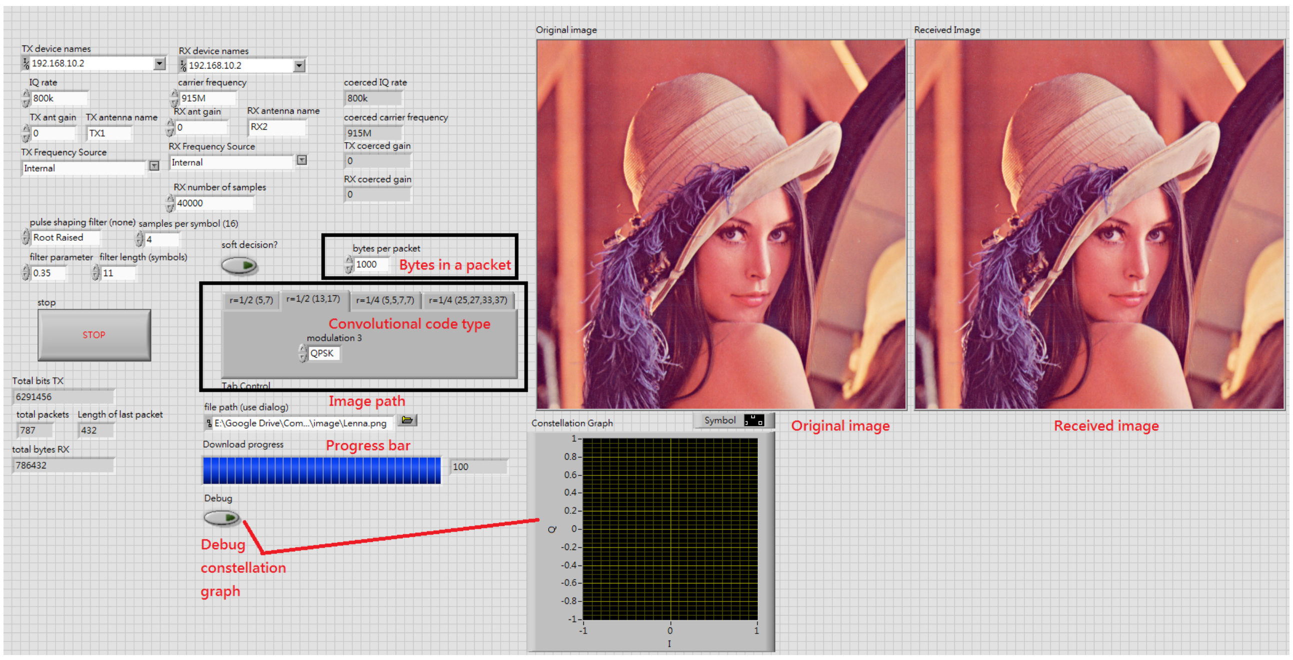

TX device names 192.168.10.2
RX device names 192.168.10.2
IQ rate 800k
carrier frequency 915M
coerced IQ rate 800k
TX ant gain 0
TX antenna name TX1
RX ant gain 0
RX antenna name RX2
coerced carrier frequency 915M
TX Frequency Source Internal
RX Frequency Source Internal
TX coerced gain 0
RX number of samples 40000
RX coerced gain 0
pulse shaping filter (none) Root Raised
samples per symbol (16) 4
filter parameter 0.35
filter length (symbols) 11
soft decision?
bytes per packet 1000
stop
Total bits TX 6291456
total packets 787
Length of last packet 432
total bytes RX 786432
r=1/2 (5,7)
r=1/2 (13,17)
r=1/4 (5,5,7,7)
r=1/4 (25,27,33,37)
modulation 3 QPSK
Tab Control
file path (use dialog) E:\Google Drive\Com...\image\Lenna.png
Download progress 100
Debug

Figure 5: Image transceiver system: Control panel

4 Exercises

4.1 LabVIEW

1. Implement the encoders and decoders (Viterbi algorithm) of [13,17] and [25,27,33,37]. Try to generate random 50 bits. Then, these bits are transmitted by encoder, and recovered by decoder. Record trellis diagrams and draw state transition maps.
2. Added convolutional codes in your QPSK system. Plot BER versus $\frac{E_b}{N0}$ curves of uncoded [13,17] and [25,27,33,37] with different decision decoders in QPSK system. The curves should start from $\frac{E_b}{N0} = 0$ all the way to the $\frac{E_b}{N0}$ that generates the BER of 10^{-5}. You should have no less than 10 simulation points spreading evenly on the curve. Discuss advantages and disadvantages of coded system.

4.2 USRP

1. Try to explain the "header" of image compression. Which type assigned image has the "header" ?
2. Recover images from received message and post results in your report. Set TX gain to 0, 5, 10.
3. Find ways to evaluate picture quality. Explain whether it is reasonable.

5 Lab Report

There is no format requirements for your lab report. In the report, you should address the results of the exercises mentioned above. You should also include your simulation program in the appendix of the report. Include whatever discussions about the new findings during the lab exercise, or the problems encountered and how are those solved. Do not limit yourself to the exercises specified here. You are highly encouraged to play around with your simulation program on self-initiated extra lab exercises/discussions.