DTSim

Data Transmission Simulator

Josep Famadas Alsamora

1. Introduction

When it comes to transmitting a message, there are two main issues that concern both the transmitter and the receiver: The reliability and the security.

1.1. Reliability

We want the message to arrive to the destiny without any error so when it is received after the channel the receiver looks for any possible error. If there is any, it tries to fix it (Forward error correction FEC), and if it is not possible it asks for the repetition of the transmission (Automatic Repeat reQuest ARQ).

The technique used to increment the reliability of the transmission is known as channel coding, which consists on adding some redundancy to the message in order to be able to use that to know whether the message has been received correctly or not.

1.2. Security

If, by any chance, an undesired third person manages to get the message we don’t want him or her to be able to know its contain. What we do is: Instead of transmitting it clearly, we modify the message in a secret way that only the transmitter and the receiver know, so if an attacker gets the message he will only read what seems to be random symbols. This technique is known as cryptography.

The Data Transmission Simulator (DTSym from now) is, as its name suggest, a simulator with which the user can load or write a message and navigate through the whole process of encryption, channel coding, transmission, channel decoding and decryption.

2. Theoretical Background

There are several ways of channel coding and encryption but in this project, I have focused in one of each. For the channel coding part I have used a convolutional code and for the cryptography part I have used the Vigenère algorithm.

2.1. Convolutional Codes

First of all, let's make a brief introduction on the math's that explain the channel coding.

Having a user word X, with a size of *k* bits, a linear function is applied on it to get the code word Y, with a size of *n* bits, being n > k.

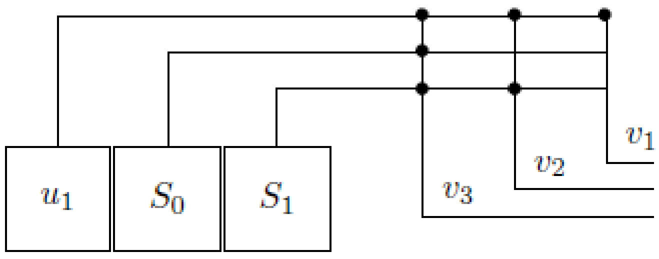
$$Y = f(X)$$

Convolutional codes are lineal codes but, unlike block codes, they have memory, which means that the code word bits do not only depend on the corresponding user word bit but also on the previous ones.

$$y(i) = f(x(i), x(i-1), x(i-2), \dots)$$

Another characteristic is that for every *n* input bits, *m* output bits are generated.

An example of convolutional code is this one with n = 1 and m = 3:



It starts with S0 and S1 both 0, and u1 = x(1), and every time the outputs are computed the next input enters the function. The second time will be u1 = x(2), S0 = x(1) and S1 = 0, and so on until u1 = 0, S0 = 0 and S1 = x(k).

In this case the functions would be:

$$v_1 = u_1$$
$$v_2 = u_1 + S_1$$
$$v_3 = u_1 + S_1 + S_0$$

The decoding of the convolutional code consists on using the Viterbi Algorithm, which will be explained in the final part of the work.

2.2. Vigenère algorithm

The Vigenère algorithm is a method of encrypting a text by using the method of polyalphabetic substitution. It consists on creating an alphanumeric key, known by the receiver and the transmitter, which is added word by word periodically to the clear text.

The mathematical function for each symbol of the cryptogram is:

$$C(i) = \big(M(i) + K(i)\big) mod(N)$$

Where M is the clear message and K is the key.

N is the size of the alphanumeric group you are using, for example, for the alphabet it will be 26. We will be using the extended ASCII code so N = 256.

This is an example of Vigenère algorithm where the clear message is “Attack at dawn” and the Key is “Lemon”, using N = 26.

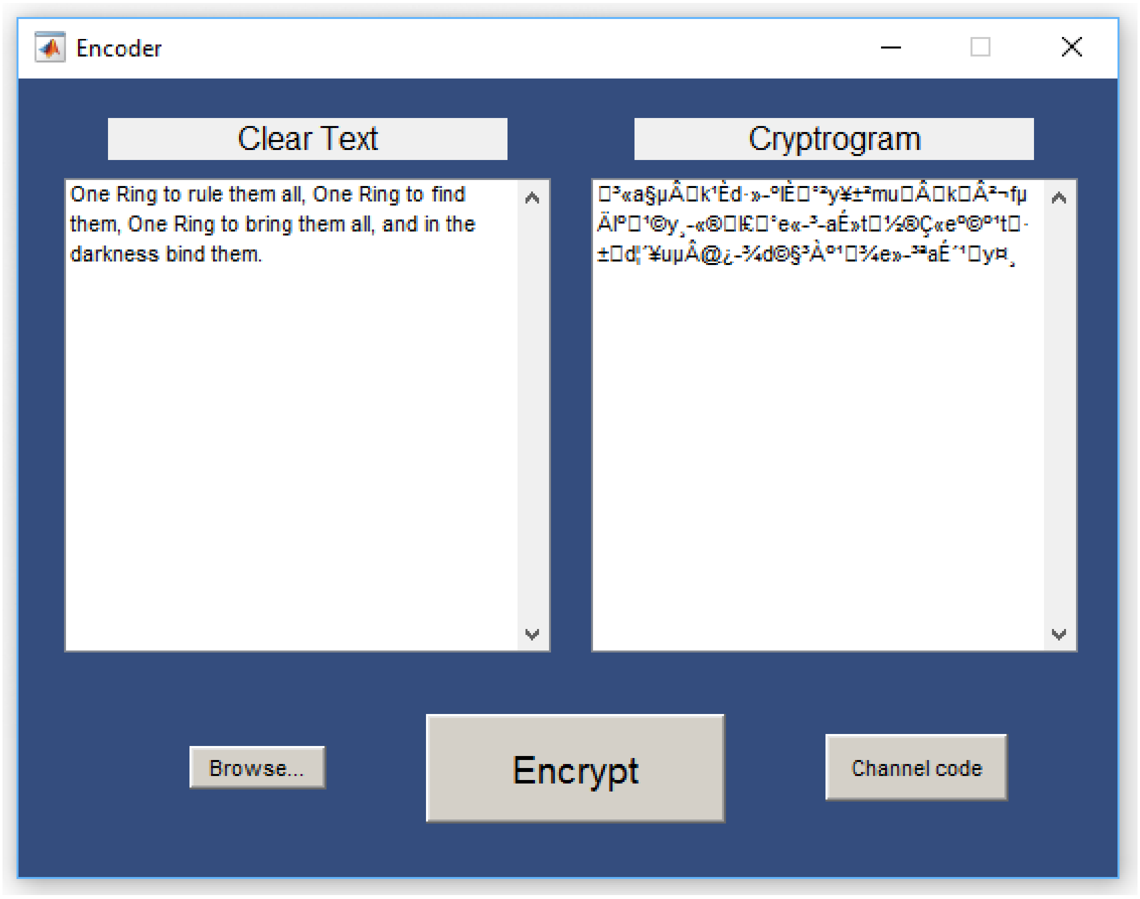
```
Plaintext:  ATTACKATDAWN

Key:        LEMONLEMONLE

Ciphertext: LXFOPVEFRNHR
```

3. Code Structure

The code consists in 4 GUI:



```
One Ring to rule them all, One Ring to find
them, One Ring to bring them all, and in the
darkness bind them.
```

```
□³«a§µÂ□k¹Èd·»-ºÌÈ□°²y¥±²mu□Â□k□Â²¬fµ
ÃÌº□¹©y¸-«®□Ì€□°e«-³-aÉ»t□½®Ç«eº©º¹t□-
±□d¦˜¥uµÂ@¿-¾d©§³Àº¹□¾e»-³ªaÉ´¹□y¤¸
```

Encoder:

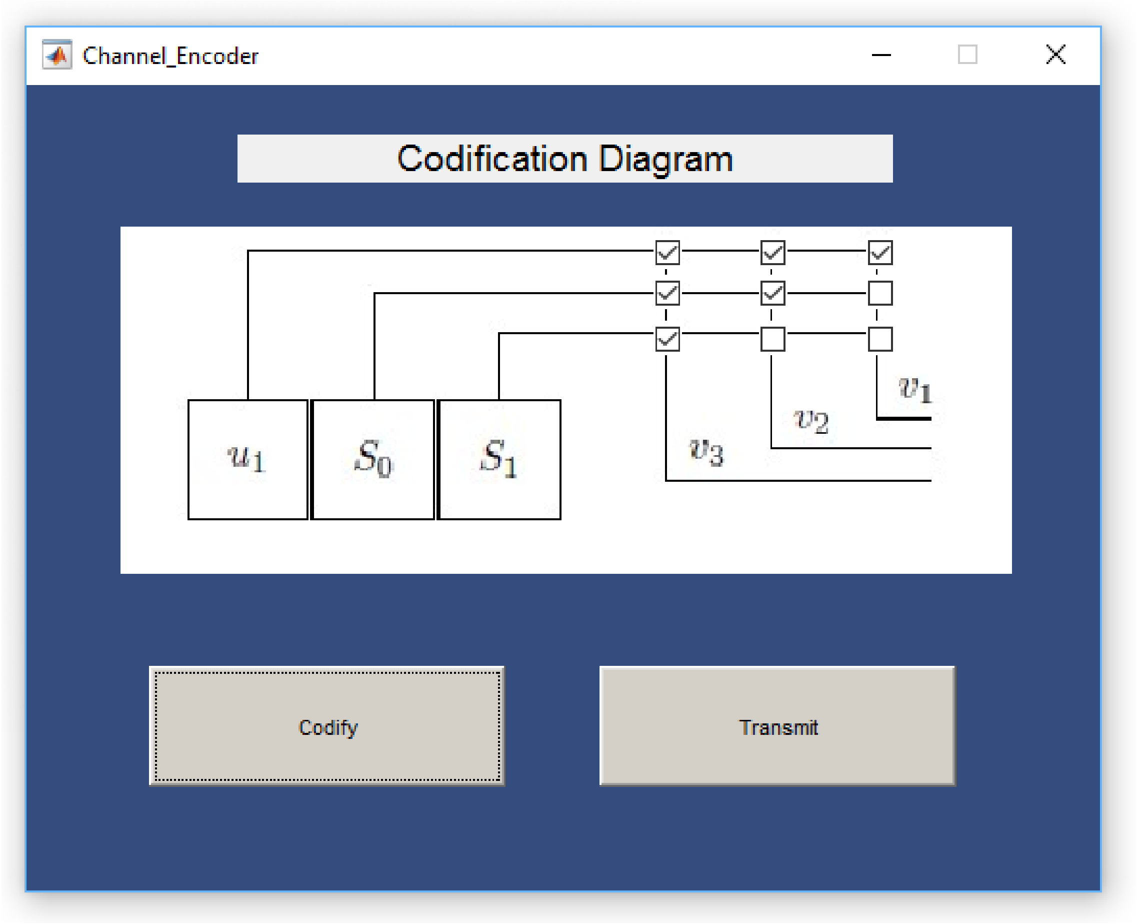
It is the first GUI, it appears when the program is executed. It consists on:

-Two text displays, the left one for the clear message and the right one for the cryptogram.

-The 'Browse...' button with which you can load a '.txt' file from your computer.

-The Encrypt button which takes the left display text, calls the created function 'Vignere_encipherment' with the text as an input parameter, and gets the cryptogram, writing it in the right display.

-The 'Channel code' button which opens the 'Channel_Encoder' GUI figure with the cryptogram as input parameter.



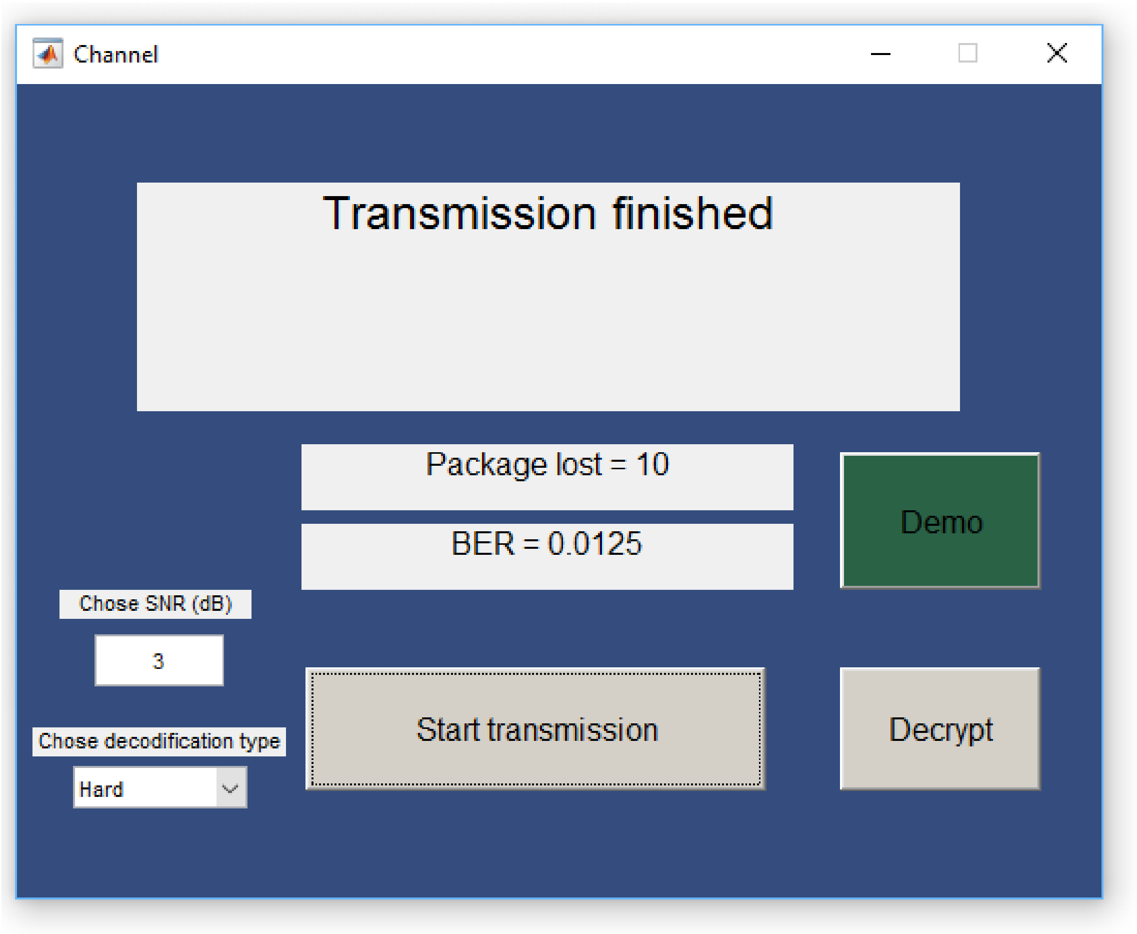
<u>Channel_Encoder:</u>

It consists on:

-A picture of a convolutional codification diagram where, by means of 9 checkboxes, the user can select which “connections” between inputs and outputs does he wants. The code length is fixed to 3, so u1 and both S must be assigned to at least one output.

-The ‘Codify’ button which uses the defined diagram to build a trellis and codify the whole sequence character by character using the created function’Channel_coder’.

-The ‘Transmit’ button which opens the ‘Channel’ GUI figure with the coded sequence, the clear sequence and the trellis vectors as input.



Transmission finished

Package lost = 10

BER = 0.0125

Chose SNR (dB): 3

Chose decodification type: Hard

Channel:

It consists on:

-A text display that varies between “Waiting”, “Transmitting… (percentage of transmission)%” and “Transmission Completed”.

-A text display with the number of packages that have need a retransmission.

-A text display with the Bit error rate.

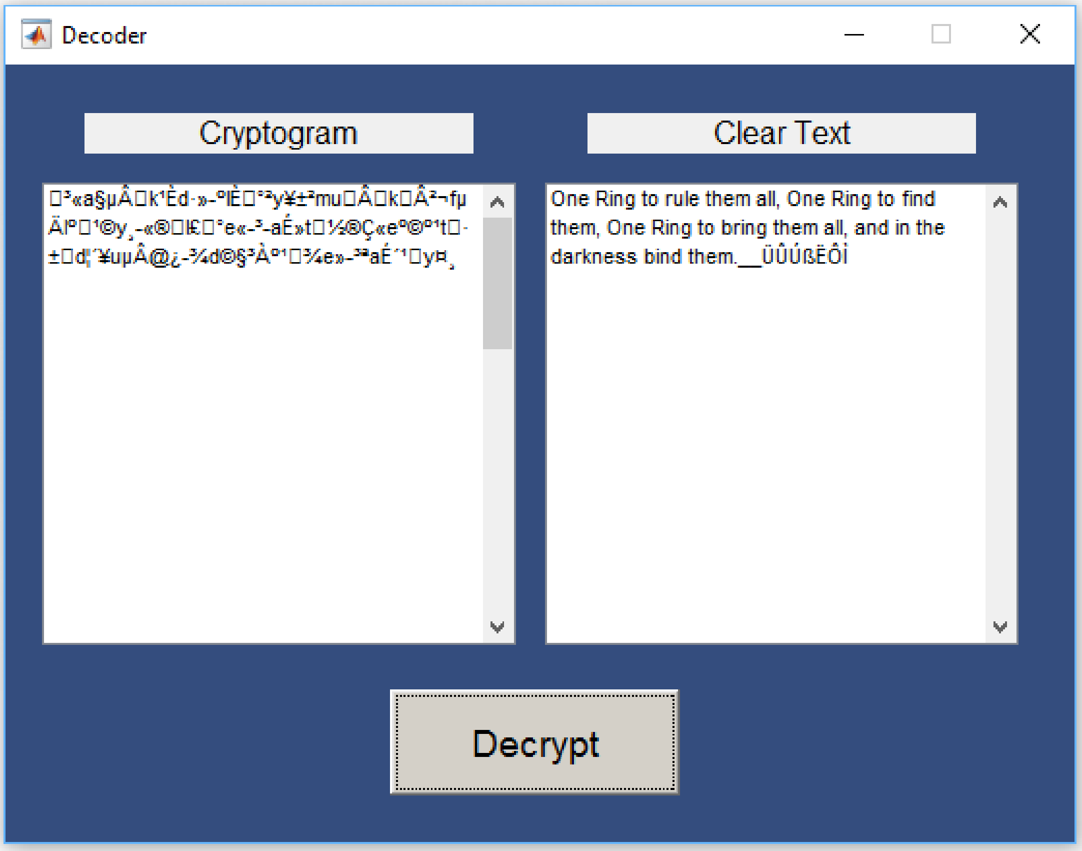
-An editable text with which the user can choose the SNR of the transmission.

-A popup menu with which the user can select the decodifying method.

-A button that transmits the packages of size one by one and retransmits them if needed. The decoding function is in this GUI, the reason is explained at the comments section.

-The ‘Demo’ button that simulates the BER in function of the SNR with an uncoded channel, with a hard decision and with a soft decision. (If the message is quite long it can take a few time).

- A last button that calls the Decoder GUI figure with the received text as a parameter.



```
□³«a§µÂ□k¹Èd·»-ºlÈ□°²y¥±²mu□Â□k□Â²¬fµ
ÃIº□¹©y¸-«®□I£□°e«-³-aÉ»t□½®Ç«eº©º¹t□·
±□d¦ˆ¥uµÂ@¿-¾d©§³Àº¹□¾e»-³ªaÉˆ¹□y¤¸
```

```
One Ring to rule them all, One Ring to find
them, One Ring to bring them all, and in the
darkness bind them.__ÜÛÚßÈÔÌ
```

Decoder

It consists on:

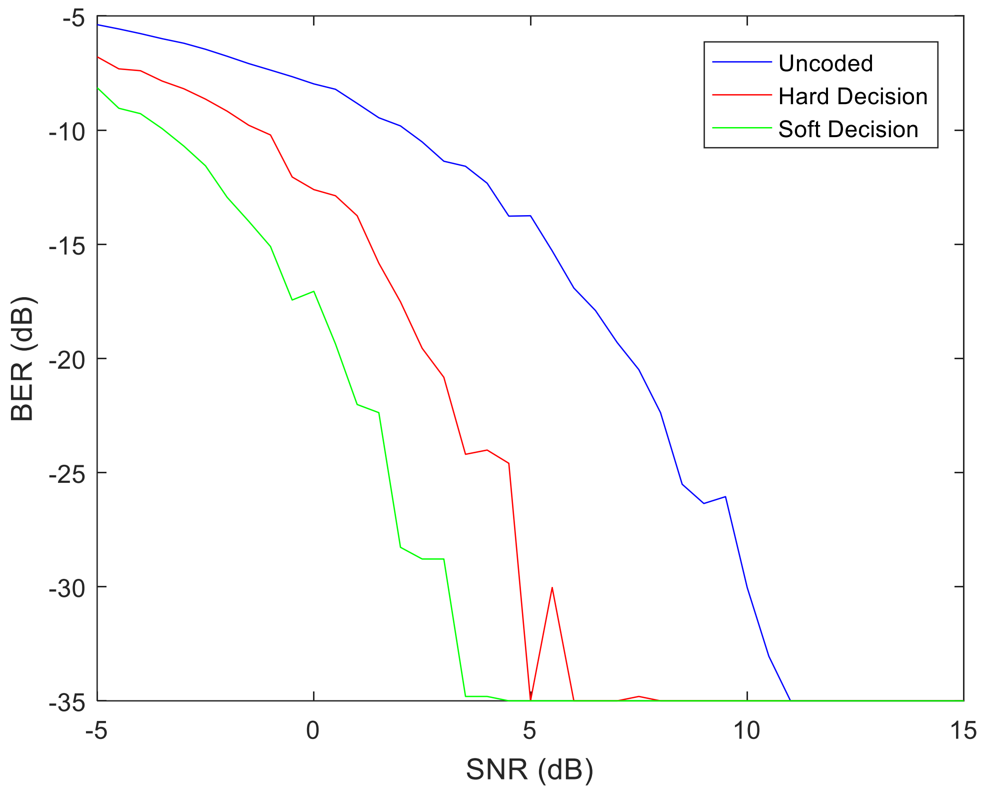
-Two text displays, the left one for the cryptogram and the right one for the clear message.

-The Decrypt button which calls the “Vigenre_decipherment” function to get the clear message from the cryptogram.

4. Results

The DTSim is made to be used, so not many results can be shown in pictures.

The following graphic corresponds to the relation between BER, SNR and the type of transmission. It is made using the pushbutton 'Demo' with a text of 600 characters. It lasts 30 seconds.



As we can expect, the uncoded transmission has far higher BER, and the Soft decodification is better than the Hard one.

5. Comments

Once the final work has been finished, I see back to the beginning of the course and realize how much have I learnt about Matlab, but looking on what could I have done but I haven't known how I also realize how much I have to learn yet.

Focusing more on this work, I have done more or less what I expected at the beginning. However, it is true that it has been something that has resisted to me. I tried in many different ways to separate the Channel GUI from the Channel Decoder function, but I was not able.

I wanted to make the ACK/NACK protocol so I would have needed a communication link between the two GUI. That might have been solved by sharing some variables through the root with the functions of 'appdata', but I found another impediment, the fact that I needed multithreading in order to keep running both loops. I tried to do it, but the information I gathered was not enough to learn it.

To sum up, despite the fact that I have not been able to do exactly everything I wanted to, I am very proud of the work I have done.