Proyecto 1: Codificación de fuente

Alejandro Chacón Vargas, B61871
Mónica Herrera Arguedas, B23253
Marlon Lazo Coronado B43717
IE-0527 Ingeniería de Comunicaciones
Escuela de Ingeniería Eléctrica, Universidad de Costa Rica
día 20 del mes de setiembre de 2021
alejandro.chaconvargas@ucr.ac.cr
monica.herreraarguedas@ucr.ac.cr
marlon.lazo@ucr.ac.cr

I. Introducción

Se procede a simular en el lenguaje de programación Python, el proceso de codificación de un canal, desde su codificador, pasando por un canal simétrico binairio, terminando con un decodificador.

II. Código en python

Primero se tomó parte del código utilizado en el proyecto anterior como se observa en la figura 2. En esta sección, se abre el fichero mensaje que posee un texto. Posteriormente los caracteres del texto se transforman en binario, luego para mayor facilidad, se transforma cada numero binario por código en 6 bits y se crea "bits a enviar"que es el mensaje que se desea transcribir pero en binario. Estos segmentos corresponden a los vectores $\vec{m}$, los cuales ya están codificados en ACSII y se pasaran por la matriz G para reducir en lo posible los errores de transmisión en los bits, al pasar estos por el canal. También cabe mencionar que los caracteres que se leen en el archivo, se transforman a binario, y se obtiene una cadena de bits, la cual luego es separada con espacios vacíos para luego convertirla en un vector de bits.

```
"""
Universidad de Costa Rica
IE0527 - Ingeniería de Comunicaciones
Proyecto 2: Codificación de canal
"""
import numpy as np

#Con esto se lee el archivo
fichero = open("mensaje.txt", "r")
#Se lee caracter por caracter
caracter = fichero.read(1)
bits_a_enviar = "" #arreglode bits a enviar
while caracter != "":

    #Transformamor a binario
    convertido_a_binario = ' '.join(format(ord(c), 'b') for c in caracter)
    #Completamos los 7 bits que a los que faltan
    #Con esto y todos los caracteres tienen la misma cantidad de bits
    while len(convertido_a_binario) <= 6:
        convertido_a_binario = "0" + convertido_a_binario
    #Concatenamos todos los caracteres
    bits_a_enviar = bits_a_enviar + convertido_a_binario
    caracter = fichero.read(1)
print("The Binary Representation is:", bits_a_enviar)

bits_a_enviar = bits_a_enviar.replace('1', ' 1 ')
bits_a_enviar = bits_a_enviar.replace('0', ' 0 ')
bits_a_enviar = bits_a_enviar.split()
for i in range(len(bits_a_enviar)):
    bits_a_enviar[i] = int(bits_a_enviar[i])
```

Figura 1. Procesamiento inicial del mensaje.

Se eligió una matriz G con unas dimensiones de 6 * 12, debido a las características del mensaje. Se creó una función que genera una matriz aleatoria G que se encuentra compuesta por la matriz identidad y la matriz de paridad, como se observa en la figura **??**. Mediante la función de **numpy identity** se creó una matriz de tamaño **n**. Ahora para los bits de paridad se decidió utilizar una matriz con bits aleatorios mediante la función **random.choice de numpy**, la cual es concatenada con la matriz identidad utilizando la función **np.hstack** para obtener una matriz **G** generalizada. Mediante este método devolvemos la matriz **G** y la matriz de paridad **P** para reutilizarla en la decodificación.

```
#====================LISTO EL ARREGLO DE BITS==========================
#====================CODIFICACIÓN Y CANAL==============================
def make_G(n,m_n): #Se crea la matriz G,  de 6x6
    I = np.identity(n)
    P = np.random.choice([0, 1], size=(n,m_n),p=[1./2, 1./2])
    G = np.hstack((I, P))
    print("La matriz G es:\n", G)
    return G, P
n = 6
G, P = make_G(n, 6)
```

Figura 2. Creación de la matriz G.

Se realizó una función que es la encargada de tomar los bits de la etapa anterior y dividirlos en los vectores m que contengan la misma cantidad de bits que el número de filas de la matriz G. Además, una vez obtenidos se calcularon los vectores u a partir de la multiplicación de los vectores m con la matriz G. El vector y que se encuentra comentado en el código, se utilizó para probar el código previamente con mayor facilidad. El código utilizado para esta etapa es el que se observa en la figura 3.

Para esto lo que se hace es tomar al vector $\vec{M}$ del código y recorrerlo mediante un for, para generalizar la segmentación de los vectores $\vec{m}$, se utiliza el parámetro $\vec{n}$ que corresponde al de la matriz identidad, de este bucle se obtienen el vector $M_t\vec{e}mp$ de vectores $\vec{m}$. Luego mediante un bucle se recorre $M_t\vec{e}m$ y utilizando la función $np.\vec{d}ot$ se multiplica cada vector $\vec{m}$ por la matriz G, y se obtienen el vector de vectores $\vec{u}$.

```python
def vectores_m(M, G, n): #Se crean los vectores u
    u = []
    m_i = []
    M_temp = []
    cont = 1
    for i in range(len(M)):
        m_i.append(M[i])
        if cont == (n):
            M_temp.append(m_i)
            m_i = []
            cont = 0
        cont = cont + 1

    for i in range(len(M_temp)):
        T = np.dot(M_temp[i],G)
        for i in range(len(T)):
            if (T[i] % 2) == 0:
                T[i] = 0
            else:
                T[i] = 1
        u.append(T.tolist())
    #print("Los vectores u son:\n", u)
    return u , M_temp
#y = [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

u, m = vectores_m(bits_a_enviar, G, n)
```

Figura 3. Creación de los vectores m.

De manera similar a como se creó G, se realizó un código que utiliza la matriz identidad y la matriz aleatoria de paridad utilizada en la matriz G, para crear la matriz H como se observa en la figura 4.

```python
#Se crea la matriz H
def make_H(n,m_n, P):
    I = np.identity(n)
    H = np.vstack((P, I))
    print("La matriz H es:\n", H)
    return H
```

Figura 4. Creación de la matriz H

Se creó un canal binario simétrico. Este canal lo que hace es que por medio de dos for recorre los vectores u y de manera aleatoria inserta errores dentro de los mismos. Para este código, se utilizó una probabilidad de error muy baja (0.100) para que la cantidad de errores que dicho canal ingresara al mensaje original no fueran muchos como se observa en la figura 5

Este error lo diseñamos de una forma muy fácil, recorrimos el vector de vectores $\vec{u}$ mediante un bucle doble, luego mediante una función $np.\vec{random}$ con un valor de variación de $[0, 100]$, y escogimos un numero cualquiera que tendrá una probabilidad de aparición de 1 %, cada bit es pasado por un condicional que dependerá de la aparición de este número y le ingresará el error.

```python
def error(u):
    error = 0
    verdadero = 0
    for vec_i in range(len(u)):
        for bit_u in range(len(u[vec_i])):
            sorteo = np.random.randint(0,100)
            bit = u[vec_i][bit_u]
            #print("SORTEO:", sorteo)
            if sorteo == 7:
                if bit == 1:
                    bit = 0
                else:
                    bit = 1
                error = error + 1
            else:
                bit = bit
                verdadero = verdadero + 1
            u[vec_i][bit_u] = bit
    return u

u_error = error(u)
comp = np.array_equal(u_error, u)
H = make_H(6, 6, P)
```

Figura 5. Creación del canal binario simétrico

Una vez pasa por el canal, el mensaje llega al decodificador. Al decodificador le llegan los vectores v (para nuestro código por facilidad se mantuvieron con el mismo nombre u). Para verificar si los vectores poseen errores, se calculan los vectores U (vistos en clase como s), se multiplican los vectores por H y después esos vectores se comparan de manera que si son es un vector de ceros (es un vector de ceros) entonces imprime True, y si tiene un error (el vector resultante no es un vector de ceros) imprime False. El código utilizado es el que se observa en la figura 6.

```python
def make_U(u, H):
    U = []
    for i in range(len(u)):
        T = np.dot(u[i],H)
        for i in range(len(T)):
            if (T[i] % 2) == 0:
                T[i] = 0
            else:
                T[i] = 1
        U.append(T)
    return U
#Aqui se hace una comparación con los vectores V
def comparison(u, m, n):
    comp = []
    for i in range(len(u)):
        temp = u[i]
        comp = np.array_equal(temp[:n], m[i])
        #print(comp)
    return comp

make_U(u, H)
comparison(u , m , n)
```

Figura 6. Creación del canal binario simétrico

En la última parte del código se requería que los números fueran strings para que el código realizado en el primer proyecto pudiera procesar la información. Por esto se creó una función intermedia que lo que hace es transformar todo a una string como se observa en la figura [**?**].

```python
#Aqui los vectores u obtenidos pasan a ser string para entrar en la
#parte de decodificación.
def u_string(u):
    u_string = ''
    for vec in range(len(u)):
        for bit in range(len(u[vec])):
            u_string = u_string + str(int(u[vec][bit]))
    #print("EL STRING ES:", u_string)
    return u_string
```

Figura 7. Transformación a string

Para poder terminar de decodificar el mensaje, es importante recordar que el mensaje transmitido se obtiene separando los bits correspondientes al mensaje de la fuente de los bits agregados por la matriz de paridad **np.hstack**, lo cual se logra recorriendo la cadena de bits **bits_a_enviar1** del código y creando una nueva cadena que tome los primeros 6 bits de cada 12 bits, esto es porque la matriz **G** genera 12 bits de a partir de 6. En la siguiente imagen se puede ver la función generalizada que realiza esta tarea.

```python
bits_a_enviar1 = u_string(u)
#Aqui agarramos solo los primeros 6 bits del vector u que tenemos,
#ya que el tama o esta de 12 bits.
def extrac_bits(bits_a_enviar1):
    cont = 0
    m_get = ''
    for bit in bits_a_enviar1:
        if cont < 6:
            m_get = m_get + bit
            cont = cont + 1
        else:
            if cont < 11:
                m_get = m_get
                cont = cont + 1
            else:
                cont = 0
    return m_get

bits_a_enviar1 = extrac_bits(bits_a_enviar1)
```

Figura 8. Separación de los vectores $\vec{m}*$ de los vectores $\vec{s}$

Por último se reutiliza la función realizada en la primera parte del proyecto, la cual toma un string de bits y genera

el archivo tomando segmentos de bits de tamaño 7 los cuales representan el mensaje en ASCII, y va escribiendo cada carácter decodificado en el archivo mediante la función **file.write**. Esta función se puede ver en la siguiente imagen.

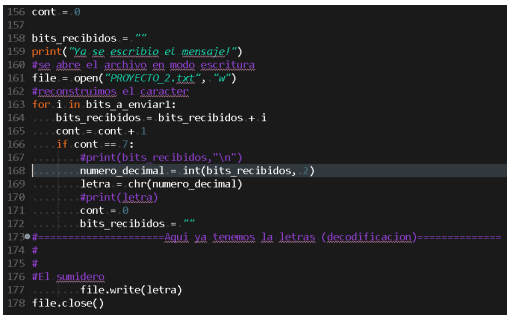

Figura 9. Función que extra los caracteres ASCII y crea el archivo

Después de todo este proceso se puede transmitir el siguiente mensaje:

Este es un mensajes de prueba
para el proyecto 1 del curso de
comunicaciones.

Este mensaje puede ser modificado, y el reseptor puede ser
borrado para recibir un nuevo mensaje.

Solo esta probado contexto sencillo, no se sabe sencillo funciona con caracteres especiales.

Y se obtiene el siguiente mensaje:

Este para el proyecto 1 del curso de gomunicaciones.
Este me saje puede ser modificado, y el resepdor puede ser
borrado
para recibir un nu % o"men°aje.

Solo esta probado Contexto‘sencillo, no se sabe sencil
o
fu ciona con caractdres eqpeciales

Claramente el segundo mensaje tiene caracteres erróneos debido al error de canal que se le puso a la cadena de bits de forma intencional. Se ha comprobado que si no se ingresa el error decanal, el mensaje se transmite perfectamente después de pasar por las matrices **G** y **H**. El código completo de este proyecto se puede encontrar en el siguiente repositorio https://github.com/Marlon-Lazo-Coronado/Ing_comunicaciones_segundaparte_proyec.git.