 INSTITUTO POLITÉCNICO NACIONAL 

"ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA y ELÉCTRICA UNIDAD

CULHUACAN"

INGENIERÍA EN COMPUTACIÓN ACADEMIA DE INGENIERÍA Y SOCIEDAD

Práctica 5: “PCM”

ASIGNATURA: Modulación Digital.

Profesor: Cruz Ramos Clara.

Equipo 1

Alumnos:

Martínez Hernández Juan Manuel.

Pachuca Vázquez Mario Alberto.

Piña Vargas Edgar Diego.

GRUPO: 6CM21

Link al documento colaborativo: <https://correoipn-my.sharepoint.com/:w:/g/personal/jmartinezh2011_alumno_ipn_mx/EX5jxccABeVLm4m_YJ-rfp4BPfwYMB14X3abTT416DfiMA?e=9o7pYk>

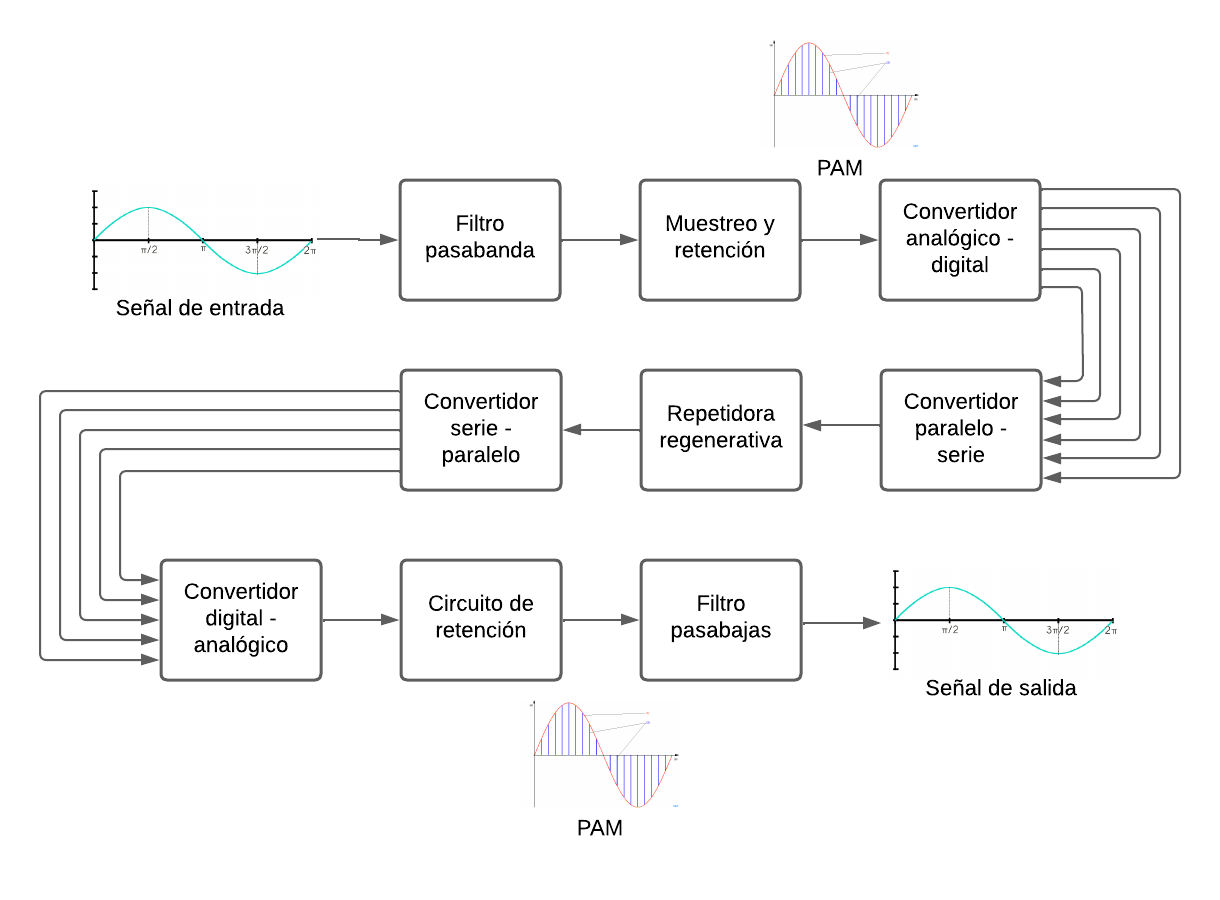
Fecha de entrega: 29/05/2024

Objetivo general.

Comprender y aplicar los principios de la modulación por codificación de pulsos (PCM) mediante la implementación de programas en Python que permitan la manipulación de imágenes y audios.

Desarrollo.

Se realizarán dos programas para PCM, donde se busca analizar una señal de audio y convertirla a mono donde veremos las características y diferencias entre ellas, también se busca leer una imagen en diferentes escalas (grises, color, blanco y negro) para verificar sus diferencias y tipos de datos.



*Diagrama a bloques de un modulador PCM*

El modulador PCM (Modulación por Codificación de Pulsos) convierte señales analógicas en digitales mediante muestreo, cuantificación y codificación. Primero, la señal analógica se muestrea a intervalos regulares, luego, cada muestra se aproxima al valor más cercano dentro de un conjunto de niveles finitos (cuantificación). Finalmente, estos valores cuantificados se codifican en secuencias de bits binarios, permitiendo su transmisión o almacenamiento digital.

Códigos.

(CÓDIGO IMAGEN)

Import tkinter as tk

From tkinter import simpledialog, messagebox, ttk

From PIL import Image, ImageTk

Import numpy as np

Class ImageApp:

Def \_\_init\_\_(self, master):

Self.master = master

Master.title(“Visor de Imágenes”)

# Cambiar el color de fondo del marco principal

Master.configure(bg=’#7D3711’) # Color “Brown”

# Crear un marco para los controles y cambiar su color de fondo

Self.controls\_frame = tk.Frame(master, bg=’#8ª5C24’) # Color “Brown”

Self.controls\_frame.pack(side=tk.LEFT, padx=10, pady=10, fill=tk.Y)

# Crear un marco para la imagen y cambiar su color de fondo

Self.image\_frame = tk.Frame(master, bg=’#FFFFFF’) # Color “White”

Self.image\_frame.pack(side=tk.RIGHT, padx=10, pady=10, fill=tk.BOTH, expand=True)

# Ruta de la imagen

Self.image\_path = “Imagen.jpg”

# Cargar la imagen original

Self.original\_image = Image.open(self.image\_path)

# Redimensionar la imagen para que sea más pequeña y conservar una copia de la original

Self.resized\_image = self.original\_image.resize((800, 550))

Self.original\_resized\_image = self.resized\_image.copy()

# Convertir la imagen a un formato que Tkinter pueda mostrar

Self.photo = ImageTk.PhotoImage(self.resized\_image)

# Crear un widget de etiqueta para mostrar la imagen

Self.label = tk.Label(self.image\_frame, image=self.photo)

Self.label.pack()

# Crear una etiqueta en la parte superior del marco de controles

Self.label2 = tk.Label(self.controls\_frame, text=”Tipo de imagen”, bg=’#8ª5C24’, font=(‘Cascadia Code ExtraLight’, 24))

Self.label2.pack(side=tk.TOP, anchor=’nw’, pady=5)

# Crear un botón para convertir a color

Self.btn\_color = tk.Button(self.controls\_frame, text=”A color (3 Matrices de 255 niveles)”, command=self.show\_original\_image)

Self.btn\_color.pack(side=tk.TOP, padx=5, pady=5)

# Crear un botón para convertir a escala de grises

Self.btn\_grayscale = tk.Button(self.controls\_frame, text=”Escala de grises (Matriz de 255 niveles)”, command=self.convert\_to\_grayscale)

Self.btn\_grayscale.pack(side=tk.TOP, padx=5, pady=5)

# Crear un botón para convertir a blanco y negro

Self.btn\_black\_white = tk.Button(self.controls\_frame, text=”Blanco y negro (Matriz de 2 niveles)”, command=self.convert\_to\_black\_white)

Self.btn\_black\_white.pack(side=tk.TOP, padx=5, pady=5)

# Crear una etiqueta para la recuantización

Self.label3 = tk.Label(self.controls\_frame, text=”Recuantización”, bg=’#8ª5C24’, font=(‘Cascadia Code ExtraLight’, 24))

Self.label3.pack(side=tk.TOP, anchor=’nw’, pady=5)

# Interfaz de la cuantizacion

Self.cuantizacionGUI = tk.Frame(self.controls\_frame, bg=’#8ª5C24’)

Self.cuantizacionGUI.pack(side=tk.TOP, padx=10, pady=10, fill=tk.X)

Self.label1 = tk.Label(self.cuantizacionGUI, text=”Cantidad bits/muestra:”, bg=’#8ª5C24’, font=(‘Arial’, 12))

Self.label1.pack(side=tk.LEFT)

Self.combox = ttk.Combobox(self.cuantizacionGUI)

Self.combox[‘values’] = (“1”, “2”, “4”, “8”) # Lista de opciones

Self.combox.current(0) # Establecer la opción predeterminada

Self.combox.pack(side=tk.LEFT, pady=10)

Self.combox.bind(“<<ComboboxSelected>>”, self.quantize\_image) # Manejar la selección

Def quantize\_image(self, event=None):

Bits = int(self.combox.get())

If bits:

If hasattr(self, ‘gray\_image’): # Si la imagen está en escala de grises

Quantized\_image = self.quantize(np.array(self.gray\_image), bits)

# Redimensionar la imagen cuantizada a las dimensiones originales

Quantized\_image\_pil = Image.fromarray(quantized\_image).resize((800, 550))

# Guardar la imagen recuantizada como archivo de imagen

Save\_filename = f”recuantized\_image\_{bits}bits.jpg”

Quantized\_image\_pil.save(save\_filename)

# Guardar el código PCM en un archivo de texto

Txt\_filename = f”recuantized\_image\_{bits}bits.txt”

Self.guardar\_binario(quantized\_image, txt\_filename, bits)

# Mostrar la imagen recuantizada

Self.photo = ImageTk.PhotoImage(quantized\_image\_pil)

Self.label.config(image=self.photo)

Else:

Messagebox.showinfo(“Información”, “La generación de PCM no está disponible en esta opción”)

If hasattr(self, ‘black\_white\_image’): # Si la imagen está en blanco y negro

Quantized\_image = self.quantize(np.array(self.black\_white\_image), bits)

Else: # Si la imagen está en color

Quantized\_image = self.quantize(np.array(self.original\_resized\_image), bits)

# Redimensionar la imagen cuantizada a las dimensiones originales

Quantized\_image\_pil = Image.fromarray(quantized\_image).resize((800, 550))

# Guardar la imagen recuantizada como archivo de imagen

Save\_filename = f”recuantized\_image\_{bits}bits.jpg”

Quantized\_image\_pil.save(save\_filename)

# Mostrar la imagen recuantizada

Self.photo = ImageTk.PhotoImage(quantized\_image\_pil)

Self.label.config(image=self.photo)

Def quantize(self, image\_data, bits):

Levels = 2 \*\* bits

Max\_val = 255 # Valor máximo para una imagen en escala de grises

Quantized = np.floor((image\_data.astype(float) / max\_val) \* levels) \* (max\_val / levels)

Return quantized.astype(np.uint8)

Def guardar\_binario(self, quantized\_image, filename, bits):

With open(filename, ‘w’) as f:

For pixel in quantized\_image.flatten():

# Convertir cada pixel a su representación binaria con el número de bits especificado

Binario = format(int(pixel \* ((2 \*\* bits) – 1) / 255), f’0{bits}b’)

f.write(binario + ‘\n’)

def convert\_to\_grayscale(self):

# Convertir la imagen a escala de grises

Self.gray\_image = self.original\_image.convert(“L”)

Self.resized\_image = self.gray\_image.resize((800, 550))

# Convertir la imagen a un formato que Tkinter pueda mostrar

Self.photo = ImageTk.PhotoImage(self.resized\_image)

Self.label.config(image=self.photo)

# Eliminar la imagen en blanco y negro si existe

If hasattr(self, ‘black\_white\_image’):

Delattr(self, ‘black\_white\_image’)

Def convert\_to\_black\_white(self):

# Convertir la imagen a blanco y negro

Self.black\_white\_image = self.original\_image.convert(“1”)

Self.resized\_image = self.black\_white\_image.resize((800, 550))

# Convertir la imagen a un formato que Tkinter pueda mostrar

Self.photo = ImageTk.PhotoImage(self.resized\_image)

Self.label.config(image=self.photo)

# Eliminar la imagen en escala de grises si existe

If hasattr(self, ‘gray\_image’):

Delattr(self, ‘gray\_image’)

Def show\_original\_image(self):

# Mostrar la imagen original a color

Self.resized\_image = self.original\_resized\_image.copy()

Self.photo = ImageTk.PhotoImage(self.resized\_image)

Self.label.config(image=self.photo)

# Eliminar la imagen en escala de grises si existe

If hasattr(self, ‘gray\_image’):

Delattr(self, ‘gray\_image’)

# Eliminar la imagen en blanco y negro si existe

If hasattr(self, ‘black\_white\_image’):

Delattr(self, ‘black\_white\_image’)

Root = tk.Tk()

App = ImageApp(root)

Root.mainloop()

(CÓDIGO AUDIO)

Pruebas y resultados.

(IMAGEN)

-Peor caso.





Recuantización.

la calidad se deteriora significativamente debido a una baja tasa de muestreo, una cuantificación poco precisa y una compresión excesiva. Esto resulta en una representación inexacta de la imagen original, con pérdida de detalles, ruido visual y distorsiones visibles. La imagen aparece borrosa, con una falta de gradaciones de gris y artefactos visibles, haciendo que la información visual sea difícil de interpretar y prácticamente inútil para su propósito previsto.

-Caso ideal.



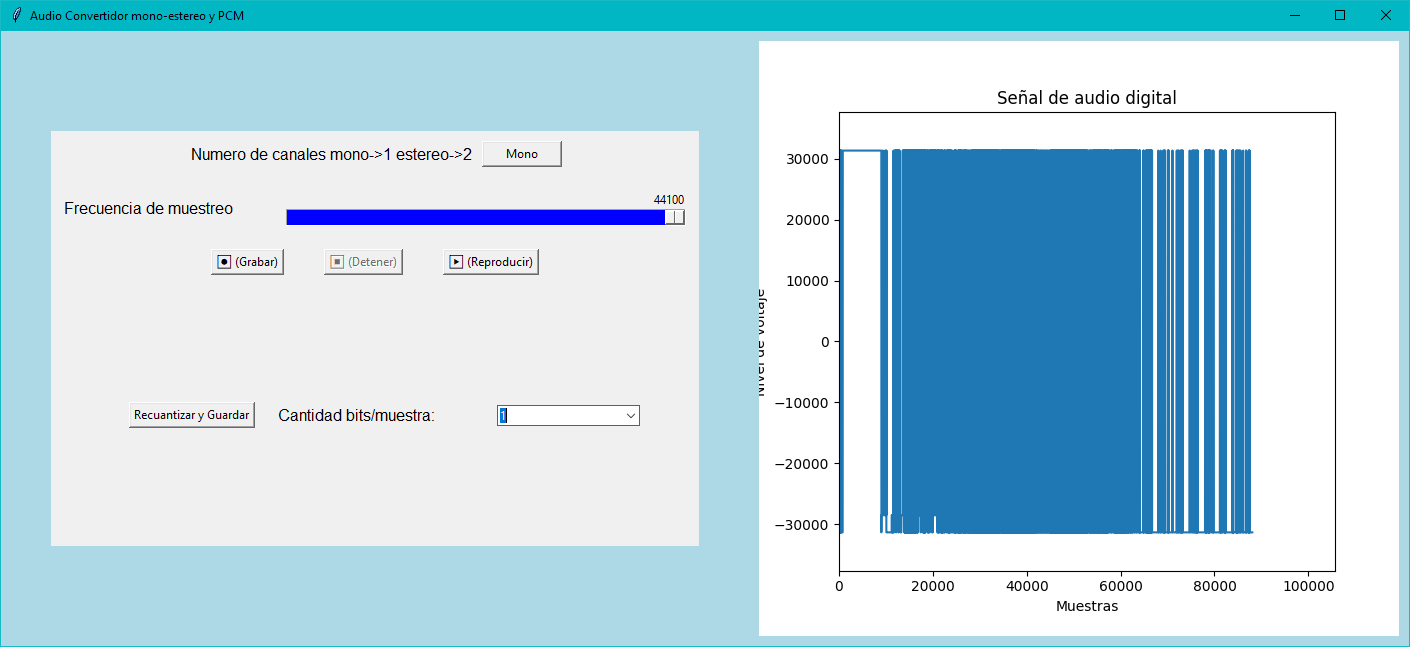


Recuantización.

Se logra una alta calidad de imagen mediante una combinación de una alta tasa de muestreo, una cuantificación precisa y una compresión eficiente. La tasa de muestreo es lo suficientemente alta como para capturar todos los detalles importantes de la imagen, mientras que la cuantificación precisa garantiza una representación fiel de los niveles de gris. La compresión es lo suficientemente inteligente como para reducir el tamaño del archivo sin comprometer significativamente la calidad visual. Como resultado, la imagen es nítida, con gradaciones suaves de gris y sin artefactos visibles, proporcionando una representación precisa y utilizable de la información visual.

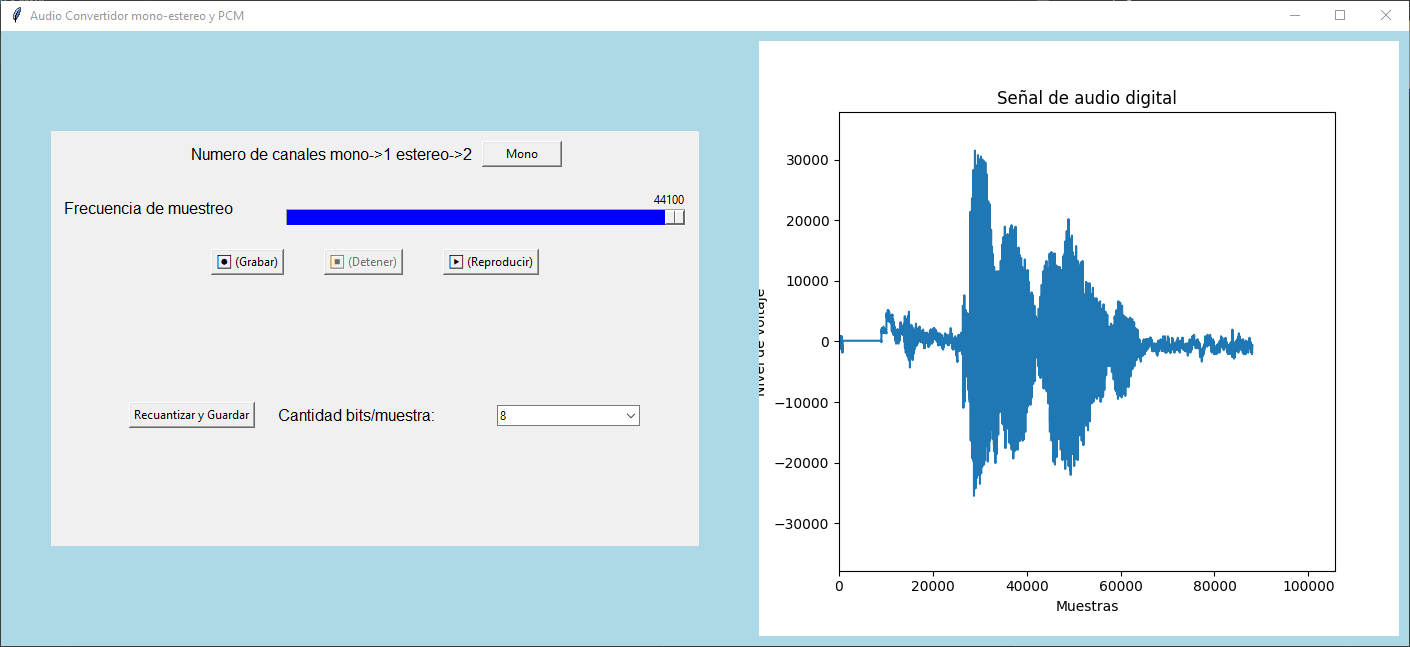
(AUDIO)

-Peor caso.



La representación gráfica muestra una señal con una calidad muy baja. Esto es debido a una tasa de muestreo insuficiente, lo que resultaría en la pérdida de detalles de la forma de onda original. Además, una cuantificación inadecuada lleva a una falta de precisión en la representación de los niveles de amplitud, lo que introduce un ruido considerable en la señal. Si a esto se le suma una compresión excesiva, la forma de onda resultante esta llena de artefactos y distorsiones, haciendo que la representación gráfica sea difícil de interpretar y poco útil para su propósito.

-Caso ideal.



La representación gráfica muestra una señal de alta calidad. Una tasa de muestreo adecuada permite capturar todos los detalles importantes de la forma de onda original, mientras que una cuantificación precisa garantiza una representación fiel de los niveles de amplitud. La compresión se aplica de manera inteligente para reducir el tamaño del archivo sin comprometer la calidad auditiva. Como resultado, la forma de onda representada gráficamente es precisa, sin artefactos visibles y fiel a la señal original, lo que proporciona una representación visual útil y precisa del audio.