Actividad Grupal 1, grupo 27, 6 (20) 10_1_21

Artefactos ScanLine. Introducción.

Las líneas de escaneo en horizontal (también conocidas como *ScanLines*) son un artefacto común en la reproducción de imágenes y vídeos en aparatos de TRC (Tubo de Rayos Catódicos, CRT en inglés), otras tecnologías Trinitrón (como algunos motores Sony BVM) y determinados monitores de videojuego arcade.
Las ScanLine son el resultado de cómo el TRC proyecta las imágenes: El tubo incluye numerosos píxeles que son dispuestos en series de arrays horizontales, cuyo número depende de la resolución vertical. Los píxeles de cada línea son iluminados a través de electrones que se desplazan por un filamento (cañón de electrones) desde la parte trasera del monitor [1].
El objetivo de la actividad es corregir los artefactos en imágenes analógicas (resultado de la fotografía de tales monitores) o digitales (aquellas que se son intencionadamente producidos con el fin de imitar a los dispositivos originales). Para este último se dispone de la señal de salida: es el caso de algunos emuladores de videojuego o animación.

Algoritmo Interpretativo-Resolutivo.

El algoritmo propuesto interpreta y analiza píxeles en sentido horizontal detectando cuándo se producen similitudes de baja iluminación (energía) en función de su repetición en sentido vertical (patrón, que definirá el salto y el grosor de las ScanLine). Es importante la detección del patrón de repetición, ya que las imágenes pueden contener espacios anómalamente oscuros que no deben interpretarse como ScanLine. Por último, cabe destacar que las imágenes analógicas a menudo muestran desaturaciones en la proximidad de las bandas que también deben de ser reconocidas y corregidas; teniendo en cuenta que las imágenes de esta naturaleza con frecuencia contienen otros artefactos que no considerará el algoritmo (efecto barril, rotaciones, etc).

```
# Importamos unas librerías básicas
import numpy as np
import matplotlib.pyplot as plt
import skimage
from skimage import io
from scipy import stats as st
import cv2
%matplotlib qt
```

1.- Cálculo de la Energía por fila y análisis de histograma

```
# Calculamos la energía como la suma de las luminancias de todos los píxeles de una misma fila.
def  Calcula_energia(imagen):
    m,n,c=imagen.shape
    r,g,b=0.2126,0.7152,0.0722
    lumi=np.uint8(np.round((r*imagen[:,:,0]+g*imagen[:,:,1]+b*imagen[:,:,2]),0))    #luminancia
    energy=np.array([0])
    for fila in range (m):
        energy =np.append(energy,lumi[fila].sum()//n)
    energy=np.delete(energy,0,0)
    return energy
```

```
# Se analiza un histograma mediante la energía de cada fila o línea, que denominaremos "histolinea"
def hist(energ,figura):
    m=len(energ)
    histolinea=np.array([[0,0]])
    for t in range (m): histolinea=np.append(histolinea,[[t,energ[t]]],axis=0)
    histolinea=np.delete(histolinea,0,axis=0)
    plt.figure(figura,figsize=(20,10))
    plt.bar(histolinea[0:m,0],histolinea[0:m,1], width=1.0,align='center')
    plt.title('Energía Líneas')
    plt.xticks(np.arange(0, m, 25))
    plt.yticks(np.arange(0, 255, 25))
    plt.show()
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return histolinea
```

2.- Detección de filas con mínimo local: Posibles ScanLine

```
'''
mediante la función detecta, devolvemos las filas que se corresponden con un mínimo local en
el array de energía; estas filas corresponderán a posibles scanlines
'''
def  detecta(energia):
    # calculamos un array con las diferencias de energía de cada fila con su anterior:
    dife=np.append(energia,[0])-np.append([0],energia)
    # para obtene los mínimos locales ponemos a 1 todos los puntos en los que la energía crece
    # y ponemos a -1 aquellos en los que la energía decrece
    dife=np.array((dife>0).astype(int)-(dife<0).astype(int))
    dife=dife+(dife==0).astype(int)*np.append([0],dife[0:-1])
    dife=dife+(dife==0).astype(int)*np.append([0],dife[0:-1])
    # Mínimos locales son puntos en los que la función pasa de -1 a 1
    # (matemáticamente de pendiente negativa a positiva)
    dife=((dife-np.append(dife[1:],[0])==-2).astype(int))
    return np.where(dife==1)[0].astype(int)
```

3.- Confirmación de ScanLine atendiendo al patrón de repetición: La función Automatica_Deteccion

Un scanline está definido por 3 características principales:

1) Una variación de luminosidad en una o más líneas horizontales.

2) Su repetición constante (o salto) a lo largo de toda la imagen cada (n) lineas, o (n, n+1) por ejemplo 7 lineas 8 líneas...

3)El grosor (g) de escanline variable, mínimo 1 línea, siempre grosor g (líneas malas consecutivas) < n (líneas de repetición de scanline).

Lo primero es detectar qué líneas responden a mínimos de luminosidad; esto se realiza buscando mínimos locales. El problema es que los mínimos pueden serlo del artefacto, pero también corresponder a mínimos correctos de la original Se necesita pues una función que discierna entre mínimos de scanline y mínimos propios a la imagen (que no hay que tratar).
La función “Automática_Detección” trata de hallar el patrón de repetición constante, y si lo encuentra, se lo devuelve a la función “detecta”. Una vez detectado corrige también posibles puntos de silla (mínimos por artefacto que no lo son matemáticamente a causa del gradiente de imagen).

```
def  Automatica_Deteccion(detecta,m,energia,pad):
    filas=detecta.shape[0]
    inicial=0 # Umbral para un porcentaje del 70% de repetición de  patron para aceptarlo
    sm=st.mode(np.append(detecta,[m])-
np.append([0],detecta)) # Busca la moda de repetición entre todos los posibles artefactos
    apariciones=sm[1][0]   # Cuántas veces se ha repetido ese valor de moda
    salto_medio=sm[0][0]   # El valor de la moda (nº de líneas en las que se repite el artefacto)
    if apariciones>=len(detectadas)*pad:    # Umbral para decidir cuántas apariciones son suficientes
        for fila in range(1,filas):         # Ha encontrado un patrón en zona donde se aprecia
            if detecta[fila]==(detecta[fila-1]+salto_medio):
                inicial=detecta[fila]       # Línea de referencia desde donde se toma el patrón
                break
        detecta1=np.array([0])
        while inicial>=0:
            inicial-=salto_medio
        inicial=inicial+salto_medio        # Primera línea de la imagen donde empezará a corregir
        for fila in range (inicial,m,salto_medio):   # Matriz automática que aplicará la corrección
            detecta1=np.append(detecta1,[fila])
        while detecta1[0]==0 :detecta1=np.delete(detecta1,0,0) # Elimina la primera scanline
    else:
        detecta1=detecta.copy()  # No ha encontrado un único patrón, se aplicará limpieza individual
    return detecta1,salto_medio
```

4.- Corrección del artefacto: la función scan_clear

Corrige las líneas en las que se ha detectado artefacto.
Hasta ahora tiene almacenado el índice de las líneas que son artefactos.
Antes de corregirlas calculará cuál es el grosor del artefacto (líneas consecutivas a ser tratadas).
Estudia las líneas anteriores y posteriores al artefacto según su variación de energía respecto al máximo valor entre artefactos consecutivos (es decir las pendientes, que a fin de cuentas son la derivada, que simplificamos por una resta).
Decide bajo ese criterio cual es el grosor izquierdo y luego cual es el grosor derecho.
El grosor total es 1 (la línea dónde se detectó scanline (mínimo)) + grosor izquierdo +grosor derecho.
Para facilitar un bucle y todos los índices, se hace un offset del inicio del scanline hasta:
detectadas[i] (índice original) -grosor izquierdo (anterior).
Cada línea nueva se genera en función de las consideradas buena anterior y buena posterior, de forma ponderada según su distancia a ambas.
La última línea es un punto singular, si tiene artefacto, sólo se le aplica una copia de la penúltima.

```
def  scan_clear(img,detectadas,ganancia):    # print("********* Rutina de limpieza \n",end='\n\n')
    m=img.shape[0]
    img_clear= img.copy()
    ganancia=1.5 # (pendientes) si valor medio es similar al maximo hay que regular a 0,5
    detectadas=np.append(np.append([0],detectadas),[img.shape[0]])
    #se usará resta matricial, se introducen 2 líneas artificiales
    for i in range(1,len(detectadas)-1):
        # Cada artefacto puede tener diferente grosor (nº de líneas malas a ser tratadas)
        grosor,grosor_izq,grosor_der=1,0,0
        #Determina si hay más lineas a izquierda y derecha que formen parte del scanline
        izq=detectadas[i-1]+np.argmax(energia[detectadas[i-1]:detectadas[i]])
        m_izq=np.mean(energia[izq:detectadas[i]])
        #Devuelve el máximo en el intervalo anterior (izquierdo)
        grosor_izq=(energia[izq+1:detectadas[i]]<ganancia*m_izq).astype(int).sum()
        #Cuántas líneas a su izquierda No llegan al umbral de buenas
        der=detectadas[i]+np.argmax(energia[detectadas[i]:detectadas[i+1]])
        m_der=np.mean(energia[detectadas[i]+1:der+1])
        #Devuelve el máximo en el intervalo anterior (izquierdo)
        grosor_der=(energia[detectadas[i]+1:der]<ganancia*m_der).astype(int).sum()
        #Cuántas líneas a su derecha No llegan al umbral de buenas
        grosor+=grosor_izq+grosor_der
        #rep es el grosor del artefacto, cuántas líneas consecutivas deben corregirse
        den=grosor+1
        fila=detectadas[i]-grosor_izq
        # Offset de la línea de inicio del artefacto según el valor a la izquierda
        for repite in range(grosor):
            #Se tratarán todas las líneas consecutivas hasta rep=grosor del artefacto actual
            if (fila+grosor)<m:
                img_clear[fila+repite]=((grosor-repite)/(den))*img_clear[fila-
1]+((1+repite)/(den))*img_clear[fila+grosor] #Corrige
            else:
                if fila+repite>=m: break  #Puntos singulares (artefacto en línea final)
                img_clear[fila+repite]=img_clear[fila-1]
#Se ha llegado al final de la matriz, se termina el tratamiento copiando la penúltima
    return img_clear,grosor
```

```
def  imprime(img,img_out,emedia, e_out,salto,grosor):
    plt.figure(figsize=(20,10))
    plt.subplot(211)
    patron=('B'*(salto-grosor)+'M'*(grosor))*2
    titulo='Imagen Original: energía media:'+str(emedia)+'  Patrón: '+ str(patron)
    plt.title(titulo)
    plt.imshow(img)
    plt.subplot(212)
    plt.title("Imagen Corregida: energía media: "+str(e_out))
    plt.imshow(img_out)
    plt.show()
```

5.- El algoritmo: Tipos de imagen detectadas y Menu Principal

A la hora de seleccionar las imágenes hemos dividido en grupos dependiendo de las características de la imagen (Se incluye anejo al final de la actividad):

· Imágenes digitales (estándar):

Este bloque de imágenes corresponde a la definición clásica de ScanLine donde aparece una franja de un pixel oscuro que se repite en intervalos de mismo tamaño (grosor 1 y salto 2). Se dispone de la señal de salida, por lo que no se combinan con otros artefactos. Son intencionadamente producidas por emuladores o reproductores. Corresponde a la imagen 1.png.

· Imágenes digitales (fondo negro):

Considerar imágenes con un fondo negro supone un desafío a la hora de evaluar el algoritmo ya que puede malinterpretar patrones de ScanLine donde no los hay. El "fondo negro" era una solución frecuente en videoconsolas con limitada capacidad de computación como la NES, equipo de 8 bits que tenía restricciones de canales de color, de sonido y de sprites (figuras) entre otras. Cada sprite sólo podía contener 4 canales de color (o 3 y uno alfa) y en cada scanline sólo podía aparecer un máximo de 8 sprites al mismo tiempo; de lo contrario se producía un fenómeno conocido como "flicker" (los sprites comenzaban a parpadear) [2]. Para evitarlo, cuando era necesario se mostraba un fondo uniforme negro, reduciendo así el número de canales por ScanLine. Corresponden a las imágenes comprendidas entre 2.png y 6.png inclusive.

· Imágenes digitales (otras variaciones):

Se considera otro bloque de imágenes en los que el efecto ScanLine no se presenta como una línea negra que se repite en función de grosor 1 y salto 2 (sino con otras variaciones). Además, se contemplan casos donde la scanline figura como una línea desaturada con respecto a las que la rodean. Corresponden a las imágenes comprendidas entre 7.png y 10.png inclusive.

· Imágenes analógicas

En este tipo de imágenes observamos que el ancho de la franja del ScanLine es variable, añadiendo en muchas de las imágenes una proximidad al mismo también oscurecida. Son imágenes analógicas las fotografías tomadas a monitores con esta particularidad. Por su naturaleza, pueden presentar otros artefactos en combinación. Corresponden a las imágenes comprendidas entre 11.png y 15.png inclusive.

```
pad, gain =0.7, 1.5
#Porcentaje automático detección (0:1)//Gain, ganancia limpieza sobre la media izquierda y derecha
img_dir = input("- Introduzca el nombre del archivo con extensión (ejemplo: 1.png)->")
img = io.imread(img_dir)
energia=Calcula_energia(img)
histolinea=hist(energia, 'histolinea IN')
detectadas=detecta(energia)
detectadas1,salto=Automatica_Deteccion(detectadas,img.shape[0],energia,pad)
img_out,grosor=scan_clear(img,detectadas1,gain)
io.imsave("img_out.png",img_out)
energia_out=Calcula_energia(img_out)
histolinea_out=hist(energia_out,'histolinea OUT')
imprime(img,img_out,energia.sum()//len(energia),energia_out.sum()//len(energia_out),salto,grosor)
```

6.- Conclusiones

· Analizando los histogramas de línea éstos muestran similitudes con una señal de radiofrecuencia analógica.

· Estudiando los histogramas de las imágenes de entrada y las corregidas, éstas últimas eliminan los peines, extrayéndose la envolvente.

· Las imágenes con menos energía corresponden a aquellas con fondos negros o gran concentración de azules y rojos, dado a que su aporte a la luminancia es bajo (7% y 21% respectivamente).

· Las imágenes con gran variación de energía desde su zona superior a su zona inferior enmascaran el scanline y dificultan la detección de mínimos locales; no obstante, el algoritmo propuesto está diseñado para detectar esta clase de imágenes.

EXTRA: Corrección de artefactos ScanLine en animaciones

La corrección de artefactos ScanLine en animaciones puede resultar de gran complejidad espacial y temporal si se evalúa el artefacto fotograma por fotograma. Una particularidad de este tipo de artefactos es que se mantiene constante en todos los fotogramas, por lo que, es factible utilizar el algoritmo Interpretativo-Resolutivo en el primer fotograma y emplear el patrón o método resultante para todos los fotogramas (sin re-evaluación, agilizando así el procesamiento).

A continuación, se muestra el algoritmo de corrección de animaciones en las que previamente se ha evaluado el primer fotograma y ya se conoce el parámetro "salto" y "grosor" (2 y 1 respectivamente para este caso).

El algoritmo de corrección de animaciones se basa en "descomposición-corrección-combinación" sin el paso interpretación que se supone dado. La corrección se agiliza empleando métodos aditivos de "solape", es decir, solapando a la imagen original una copia desplazada verticalmente en función del patrón de partida. Es fundamentalmente una operación a nivel de píxel, en función de la altura y distanciamiento de las bandas.

```
import cv2
import numpy as np
import os
import re
from PIL import Image, GifImagePlugin
#1.- DESCOMPONE EL GIF INTRODUCIDO EN FOTOGRAMAS (Imágenes .bmp)
archivo = input("- Introduzca el nombre del archivo con extensión (ejemplo: a.gif)->")
print("- Proceso en curso... Por favor, espere.")
animacion = Image.open(archivo)
for fotograma in range(0,animacion.n_frames):
    animacion.seek(fotograma)
    animacion.save(str(fotograma) + ".bmp")
#2.- CORRIGE LOS FOTOGRAMAS MEDIANTE "SOLAPE" (Según la evaluación del algoritmo genérico)
for fotograma in os.listdir():
    if fotograma.endswith(".bmp"):
        #2.1.- Capa original:
        original = cv2.imread(fotograma)
        original = np.array(original,np.int32)
        #2.2.- Capa desfasada en función del parámetro "salto" y "grosor":
        desfase = np.array(np.delete(original, 0, 0))
        desfase = np.append(desfase, np.array([desfase[-1]]), axis=0)
        #2.3.- Solape resultante:
        solape = np.add(original, desfase)
        solape = np.clip(solape, 0, 255)
        solape = np.array(np.delete(solape, -1, 0))
        #2.4.- Sobrescribe cada fotograma con la corrección
        cv2.imwrite(fotograma, solape)
#3.- COMBINA LOS FOTOGRAMAS CORREGIDOS
#3.1.- Genera una lista con todos los fotogramas png del directorio
#      los pone en orden natural para evitar orden alfabético (1,10,11..19,2,20,21):
lista=[]
for fotograma in os.listdir():
    if fotograma.endswith(".bmp "):
        lista.append(fotograma)
def texto_a_int(texto):
    return int(texto) if texto.isdigit() else texto
def texto_natural(text):
    return [ texto_a_int(letra) for letra in re.split('(\d+)',text) ]
lista.sort(key=texto_natural)
#3.2.- Guarda un gif combinando todos los fotogramas de la lista:
lista_img=[]
for n_lista, imagen in enumerate(lista):
    n_lista = Image.open(imagen)
    lista_img.append(n_lista)
lista_img[0].save("gifcorregido_"+archivo, save_all=True, append_images=lista_img[1:], loop=0)
#4.- ELIMINA LOS .bmp, LIMPIANDO EL DIRECTORIO Y DEJANDO, ÚNICAMENTE, EL GIF CORREGIDO
for fotograma in os.listdir():
    if fotograma.endswith(".bmp"):
        os.remove(fotograma)
print("- Proceso terminado. Consulte el archivo 'gifcorregido_(nombre)' en el directorio raíz.")
```

Referencia Bibliográfica

[1] G. Wiesen (17 de diciembre de 2020). *What is a Scan Line?* https://easytechjunkie.com

[2] J.McDonald. *Why Do NES Games Flicker – A Visual Explanation.* https://retrogamestart.com/

ANEJO: Selección de imágenes de estudio

Figura 1. Imágenes Digitales. De izquierda a derecha: Imagen digital estándar, de fondo negro y de variaciones desaturadas.

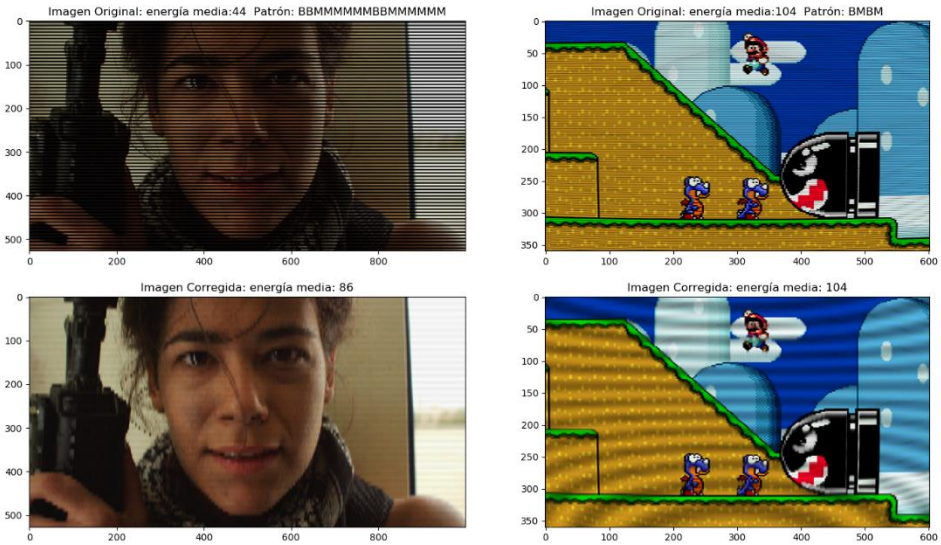

Figura 2. Imágenes Analógicas. La corrección del artefacto ScanLine desvela otros artefactos (como por ejemplo, efecto barril).

Figura 3. Corrección de Animación. Las soluciones no interpretativas (interpretación previa) agilizan el proceso de composición.