Usando conversion RGB-HSI/HSI-RGB con ejecución en paralelo para alterar color en video

May 20, 2019

1 Introducción

El procesamiento de imágenes cada vez está tomado un rol más preponderante en los campos de investigación; esto se debe a distintos factores: Mayores campos de aplicación: por ejemplo análisis de imágenes satelitales destinadas para las clasificaciones de áreas agrícolas [1]

Mejor potencia en los procesadores y memoria: Las tarjetas gráficas han mejorado de manera exponencial en la última década, su aplicación común es el procesamiento de muchos cálculos matemáticos, específicamente (en el uso doméstico) orientadas a los videojuegos. Los fabricantes de procesadores como Intel, aumentan cada vez más las prestaciones de sus productos, tanto los ordenadores personales como los mainframes. [2]

Mayor espacio en medios de almacenamiento: Cada vez las imágenes digitales aumentan su resolución, haciendo que se necesite de mayor capacidad de almacenamiento. El análisis y procesamiento del color de imágenes y videos es el foco de estudio de este documento, poniéndolo en forma sencilla:

- Seleccionara un color
- Cambiará de espacio de color
- Modificará una propiedad
- Volver al espacio de color original

2 Modelos

2.1 El modelo de color RGB

Este modelo es un subespacio del espacio euclidiano conformado por el cubo unitario. . Los colores aparecen con sus componentes primarias de rojo, verde y azul. Los valores de R, G y B se encuentran a lo largo de tres ejes. [3]

2.2 El modelo de color HSI

En el modelo de color HSI los colores se distinguen unos de otros por su tono, intensidad, y saturación. El tono está asociado con la longitud de onda dominante en una mezcla de ondas luminosas. La intensidad representa la iluminación percibida. La intensidad da la sensación de que algun objeto refleja mas o menos luz. La saturacion se refiere a la cantidad de luz blanca mezclada con el color dominante. [4]

2.3 Conversión de un espacio a otro

2.3.1 Conversión de RGB a HSI

Teniendo en cuenta [5]:

Determinar valores para canal I:

$I = \frac{1}{3}(R + G + B$

Determinar valores para canal H:

cuando G>B

$H = cos^{-1}\left(\frac{(R-G)+(R-B)}{\sqrt[2]{(R-G)^2+(R-B)(G-B)}}\right)$

cuando B>G

$H = 360 - H$

Determinar valores para canal S:

$S = 1 - 3min\left(R, G, B\right)/I$

2.3.2 Conversión desde HSI a RGB

Teniendo en cuenta [5]:

0<H<120

$b = \frac{1-S}{3}$

$r = \frac{1}{3}\left[1 + \frac{ScosH}{cos(60-H)}\right]$

$g = 1 - b - r$

120<H<240

$H = H - 120$

$r = \frac{1-S}{3}$

$g = \frac{1}{3}\left[1 + \frac{ScosH}{cos(60-H)}\right]$

$g = 1 - g - r$

240<H<360

$H = H - 240$

$g = \frac{1-S}{3}$

$b = \frac{1}{3}\left[1 + \frac{ScosH}{cos(60-H)}\right]$

$r = 1 - g - r$

3 Implementación

En simples términos, sabemos que un video no es más que una composición de imágenes, para cambiar un color en un video, debemos hacerlo para cada uno de los frames que compone el video en cuestión. La implementación que propone este artículo se compone de tres módulos desarrollados en Python y un último módulo realizado en C++ para obtener resultados más rápidos.

3.1 Módulo: Frames

El primer módulo es responsable de obtener los frames de un video. Se lo ha empleado para conocer la cantidad de frames que posee el video (dado que OpenCV no posee un buen mecanismo para obtenerlo) y para obtener una galería de imágenes con los frames.

```
# coding: utf-8
import cv2
import numpy as np
import sys

#defino el video
your_path=sys.argv[1]
your_destiniy_path=sys.argv[2]
vidcap = cv2.VideoCapture(your_path)

#calculo la cantidad de frames
total_frames = str(int(vidcap.get(cv2.CAP_PROP_FRAME_COUNT)))
#los nombres de las imagenes deben tener las misma cantidad de
    caracteres ej 000.jpg, 001.jpg, 025.jpg, 712.jpg
length= len(total_frames)
nombre=""
success,image = vidcap.read()
count = 1

try:
    while success:
        nombre=str(count)
#se la el formato al nombre
        while (len(nombre) != length):
            nombre="0"+nombre
#Se le defina la ruta completa :v
        cv2.imwrite(your_destiniy_path+nombre+".jpg", image)
        success,image = vidcap.read()
        count += 1
except ValueError:
    print(-1)
```

3.2 Conversión

El segundo módulo es el que se encarga de la transformación de los espacios de colores para un frame.

1. Pasamos del modelo RGB a HSI
2. Separamos los canales, de este modo podremos trabajar con el canal H
3. modificamos un rango de valores del canal H (por ejemplo aquellos que sean los correspondidos a un color azul)
4. Reunimos los canales del HSI
5. Pasamos del modelo HSI a RGB

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
import sys

def rgb2hsi(rgb):
    # separar
    R,G,B= cv2.split(rgb)
    # normalizar
    R =R/255
    G =G/255
    B =B/255
    # cantidad de elementos
    x=R.shape[0]
    y=R.shape[1]
    # crear arrays
    r=np.empty([x,y])
    g=np.empty([x,y])
    b=np.empty([x,y])
    H=np.empty([x,y])
    S=np.empty([x,y])
    I=np.empty([x,y])

    # recorrer
    for i in range(0, x):
        for j in range(0,y):
            # calcular rgb
            divisor=R[i,j]+G[i,j]+B[i,j]
            I[i,j]= divisor/3.0
            if (divisor != 0.0):
                r[i,j]=R[i,j]/divisor
```

```
                g[i,j]=G[i,j]/divisor
                b[i,j]=B[i,j]/divisor

            # calcular RGB
            if (R[i,j]==G[i,j]) and (G[i,j]==B[i,j]):
                H[i,j]=0
                S[i,j]=0
            else:
                argum=(R[i,j]-G[i,j])*(R[i,j]-G[i,j])+(R[i,j]-
                    B[i,j])*(G[i,j]-B[i,j])
                num=0.5*((R[i,j]-G[i,j]) + (R[i,j]-B[i,j]))
                w=num/math.sqrt(argum)
                if (w>1): w=1
                if (w<-1): w=-1

                H[i,j]=math.acos(w)
                if H[i,j] < 0:
                    print('b')
                    break

                if B[i,j] > G[i,j]:
                    H[i,j]=2*math.pi-H[i,j]

                if (r[i,j] <= g[i,j]) & (r[i,j] <= b[i,j]):
                    S[i,j]=1-3*r[i,j]
                if (g[i,j] <= r[i,j]) & (g[i,j] <= b[i,j]):
                    S[i,j]=1-3*g[i,j]
                if (b[i,j] <= r[i,j]) & (b[i,j] <= g[i,j]):
                    S[i,j]=1-3*b[i,j]

    H*=179
    S*=255
    I*=255
    hsi=cv2.merge([H,S,I])
    return hsi

def hsi2rgb(hsi):
    H,S,I = cv2.split(hsi)
    H=H/179
    S=S/255
    I=I/255
    x=H.shape[0]
    y=H.shape[1]
    R=np.empty([x,y])
    G=np.empty([x,y])
    B=np.empty([x,y])
    r=np.empty([x,y])
    g=np.empty([x,y])
    b=np.empty([x,y])
```

```
    for i in range(0, x):
        for j in range(0,y):
            if (S[i,j] >1): S[i,j]=1
            if (I[i,j] >1): I[i,j]=1
            if (S[i,j] ==0):
                R[i,j]=I[i,j]
                G[i,j]=I[i,j]
                B[i,j]=I[i,j]
            else:
                ums=(1-S[i,j])/3
                if (H[i,j]>=0) and (H[i,j]<np.radians(120)):
                    b[i,j]=ums
                    r[i,j]= 1/3*(1+(S[i,j]*np.cos(H[i,j])/np.
                        cos(np.radians(60)-H[i,j])))
                    g[i,j]=1-r[i,j]-b[i,j]
                elif (H[i,j]>=np.radians(120)) and (H[i,j]<np.
                    radians(240)):
                    H[i,j]-=np.radians(120)
                    r[i,j]=ums
                    g[i,j]=1/3*(1+(S[i,j]*np.cos(H[i,j])/np.
                        cos(np.radians(60)-H[i,j])))
                    b[i,j]=1-r[i,j]-g[i,j]
                elif (H[i,j]>=np.radians(240)) and (H[i,j]<np.
                    radians(360)):
                    H[i,j]-=np.radians(240)
                    g[i,j]=ums
                    b[i,j]=1/3*(1+(S[i,j]*np.cos(H[i,j])/np.
                        cos(np.radians(60)-H[i,j])))
                    r[i,j]=1-g[i,j]-b[i,j]
                else:
                    print("fuera de rango")
                    break
                if (r[i,j]<0): r[i,j]=0
                if (g[i,j]<0): g[i,j]=0
                if (b[i,j]<0): b[i,j]=0
                R[i,j]=3*I[i,j]*r[i,j]
                G[i,j]=3*I[i,j]*g[i,j]
                B[i,j]=3*I[i,j]*b[i,j]
                if (R[i,j]>1): R[i,j]=1
                if (G[i,j]>1): G[i,j]=1
                if (B[i,j]>1): B[i,j]=1
    rgb=cv2.merge([R,G,B])*255
    return rgb.astype(np.uint8)

your_path=sys.argv[1]
#your_destiniy_path=sys.argv[2]

frame2 = cv2.imread(your_path)
```

```
mi_rgb= cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)
# Realizar conversión
mi_hsi= rgb2hsi(mi_rgb) # RGB a HSI
h,s,i = cv2.split(mi_hsi)

# Realizar el cambio de color
h=np.where((h>=640/179) & (h<=822/179),30/179,h)
mi_hsi=cv2.merge([h,s,i])

n_rgb= hsi2rgb(mi_hsi) # HSI a RGB
plt.imshow(n_rgb)

# Guardar
nuevo_bgr=cv2.cvtColor(n_rgb, cv2.COLOR_RGB2BGR)
cv2.imwrite(your_path,nuevo_bgr)
```

3.3 Reconstrucción

El tercer módulo realiza la reconstrucción del video en base a los frames modificados, Se debe tener en cuenta que la reconstrucción de este tipo de video suele ser lenta y será optimizada por medio del cuarto módulo.

```
import cv2
import os
import sys

# Arguments
#dir_path = directorio donde se encuentran los frames
#ext = extención de los frames
#output = nombre del archivo  ruta_de_archivo/"output.mp4"
#inicio= primer frame desde el que se reconstruye el video
#fin= último frame hasta el que se construye el video

def rebuild(dir_path,ext,output,inicio,fin):
        try:
                mylistdir=os.listdir(dir_path)
                mylistdir.sort()
                mylistdir.pop(0)

                images = []
                for f in mylistdir:
                    if f.endswith(ext):
                        images.append(f)

# Determina las dimensiones
                image_path = os.path.join(dir_path, images[0])
                frame = cv2.imread(image_path)
                cv2.imshow('video',frame)
```

```
                height, width, channels = frame.shape

# Define el codec
                fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                out = cv2.VideoWriter(output, fourcc, 20.0, (
                    width, height))

                for x in xrange(inicio,fin):

                    image_path = dir_path+images[x]
                    frame = cv2.imread(image_path)

                    out.write(frame)
                    cv2.imshow('video',frame)
                    if (cv2.waitKey(1) & 0xFF) == ord('q'):
                        break

                out.release()
                cv2.destroyAllWindows()
                return True
        except:
                return False

your_origin_path=str(sys.argv[1])
your_extension=str(sys.argv[2])
your_destiny_path=str(sys.argv[3])
your_init=int(sys.argv[4])
your_end=int(sys.argv[5])


#your_origin_path="/home/stark28/python_projects/paralelo-proc
    -img/img-tafirol/"
#your_extension=".jpg"
#your_destiny_path="/home/stark28/python_projects/paralelo-
    proc-img/img-tafirol/output.mp4"
#your_init=1
#your_end=329

rebuild(your_origin_path,your_extension,your_destiny_path,
    your_init,your_end)
```

3.4 Control de ejecución

Para poder aprovechar perfectamente todas las capacidades del ordenador en donde se ejecuta, podemos paralelizar la ejecución del módulo anterior. Consideremos lo siguiente, contamos con un ordenador con 4 procesadores, entonces a cada procesador le daremos una imagen para que la transforme, en un lugar de encargarle a un solo procesador todas las imágenes. Quien gobierna la ejecución de los módulos desarrollados en Python, es un código escrito en C++,

este último es quien llama a los módulos mediante llamadas al sistema, y es el encargado de distribuir el procesamiento y realizar el paralelismo.

```
#include <omp.h>
#include <stdio.h>       /* printf */
#include <stdlib.h>      /* system, NULL, EXIT_FAILURE */
#include <string>
#include <iostream>
#include <time.h>

using namespace std;

int main (int argc, char **argv)
{
  std::string origindir = argv[1];
  std::string destdir = argv[2];
  int cantidad = atoi(argv[3]);
  std::string cant_str = argv[3];

  printf ("Checking if processor is available...\n");
  time_t mytime = time(NULL);
  char * time_str = ctime(&mytime);
  printf("Current Time : %s\n", time_str);

  std::string my_command="python get_frames.py "+ origindir +
      " " + destdir;
  system ((my_command).c_str());



  printf("enter to a parallel region\n");
 #pragma omp parallel for num_threads (4)
  for (int i=1;i<=cantidad; i++){ //modificar la costante n
      para poder hacer independiente de los frames
       std::string name = to_string(i);
       while(cant_str.length() != name.length())
       {
           name= "0"+name;
       }
     std::string my_command2="python rgb_hsv.py " + destdir +"/
         "+ name +".jpg";
     system ((my_command2).c_str());
  }

    my_command="python reconstruir.py "+ destdir +". jpg "+
        destdir+"output.mp4 1 " + to_string(cantidad); //
        cambiar este hardcode
    system ((my_command).c_str());
```

```
    time_t mytime2 = time(NULL);
    char * time_str2 = ctime(&mytime2);


    printf("Current Time : %s\n", time_str2);

  return 0;
}
```

4 Resultados experimentales

4.1 Primeros intentos con HSV

Probamos con realizarle el cambio de un valor de H (de un HSV) para una imagen simple [6]

4.2 Alterar único frame con HSV

Primero se ha probado realizando el cambio de valores usando RGB-HSV/HSV-RGB solamente usando un frame del video [7] y luego con todos los frames [8]

4.3 Alterar único frame con HSI

Prueba realizada usando la conversión RGB-HSI/HSI-RGB para un único frame [9] y para un segmento de video [10]. Se hace uso plenamente del código expuesto.

5 Bibliografia:

- [1] https://tesis.ipn.mx/bitstream/handle/123456789/14571/TESIS.pdf?sequence=1&isAllowed=y
- [2] https://www.top500.org/lists/2018/06/
- [3] "Uso del sistema HSI para asignar falso color a objetos en imagenes digitales" http://www.scielo.org.mx/pdf/rmfe/v54n2/v54n2a11.pdf
- [4] "Uso del sistema HSI para asignar falso color a objetos en imagenes digitales" http://www.scielo.org.mx/pdf/rmfe/v54n2/v54n2a11.pdf
- [5] Espacios de Color RGB, HSI y sus Generalizaciones. Alonso Paz. https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/362/1/AlonsoPeMA.pdf

- [6] https://github.com/jrodrigopuca/procesamiento-img/blob/master/dia_04_test.ipynb
- [7] Código: https://github.com/jrodrigopuca/procesamiento-img/blob/master/dia_04-HSV.ipynb
- [8] Video resultado: https://youtu.be/wPPqC1ZZzk8
- [9] Código: https://github.com/jrodrigopuca/procesamiento-img/blob/master/RGB-HSI.ipynb
- [10] Video: https://youtu.be/UI5y_p6d0z8