

Laboratorio Nº 1- Introducción a OpenCV		
Asignatura	Tópicos Especiales - Visión Artificial	Código
Profesor	José Carlos Rangel Ortiz	0756

Introducción

En este laboratorio se mostrarán los aspectos básicos y fundamentales para trabajar con la librería OpenCV. Para el desarrollo del laboratorio se utilizará el lenguaje Python. El estudiante es libre de utilizar el IDE o editor de código de su preferencia.

Dentro del contenido se emplearan de igual manera las librerías Numpy y Matplotlib, las cuales serán utilizadas a lo largo del semestre.

Fundamentos Teóricos

OpenCV es un librería *open source* para visión por computador la cual esta disponible en <http://opencv.org>. Se lanzó en 1999 por Gary Bradski mientras trabajaba en Intel. Su objetivo fue el acelerar la visión por computadora y la inteligencia artificial, poniendo a disposición de los usuarios una estructura sólida para toda persona que trabajase en este campo.

Esta librería escrita en C y C++ se puede ejecutar en Linux, Windows y Mac OS. Además, existe un desarrollo activo de interfaces para Python, Java, MATLAB y otros lenguajes. En los últimos años se ha utilizado también para aplicaciones móviles tanto Android como iOS y ha recibido mucho apoyo desde Intel y Google.

OpenCV se diseñó para la eficiencia computacional con un fuerte enfoque en aplicaciones en tiempo real. Entre sus objetivos esta el proveer a los usuarios una infraestructura de uso simple para visión artificial, la cual permita construir aplicaciones bastante sofisticadas de una manera rápida. Esta compuesta por más de 500 funciones utilizadas en visión. Dentro de estas se pueden mencionar, el tratamiento de imágenes médicas, procesos industriales, calibración de cámaras, visión estéreo, robótica, entre otras.

Como elemento fundamental para para el desarrollo de este laboratorio la Figura 1 muestra el origen de coordenadas que utiliza OpenCV para el desarrollo de sus algoritmos.

OpenCV como librería, almacena la información de color de una imagen siguiendo un orden diferente al que se acostumbra, en este caso las lo canales *RGB* imágenes se guardan en el orden *BGR* dentro de la aplicación.

Como elemento adicional, se mostrará la manera de utilizar Jupyter Notebook, este es una aplicación web *open source* que permite crear y compartir documentos que contienen código fuente, ecuaciones, visualizaciones y texto explicativo de estos. Este será el entorno con el cual

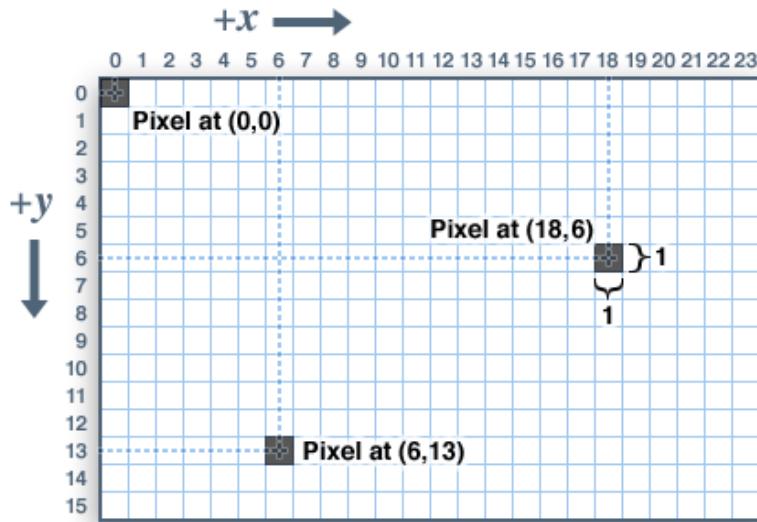


Figura 1: Origen de Coordenadas de OpenCV

se demostrarán los diferentes ejemplos durante clase, pero el estudiante puede utilizar el entorno deseado.

Objetivo

El objetivo de este laboratorio es familiarizar al estudiante con los conceptos básicos para realizar aplicaciones de visión por computador, utilizando OpenCV y el lenguaje Python.

Entregables

Al final de cada sección del laboratorio se indicará cuales son acciones que deberá cumplir el estudiante y los cuales serán evaluados en la actividad. El informe final debe contener todo lo solicitado y se debe redactar utilizando la **Guía de Estilos para Memorias y Laboratorios** Disponible en Moodle. El informe debe ser entregado en formato PDF y siguiendo las indicaciones para nombrar el archivo.

1. Introducción a OpenCV

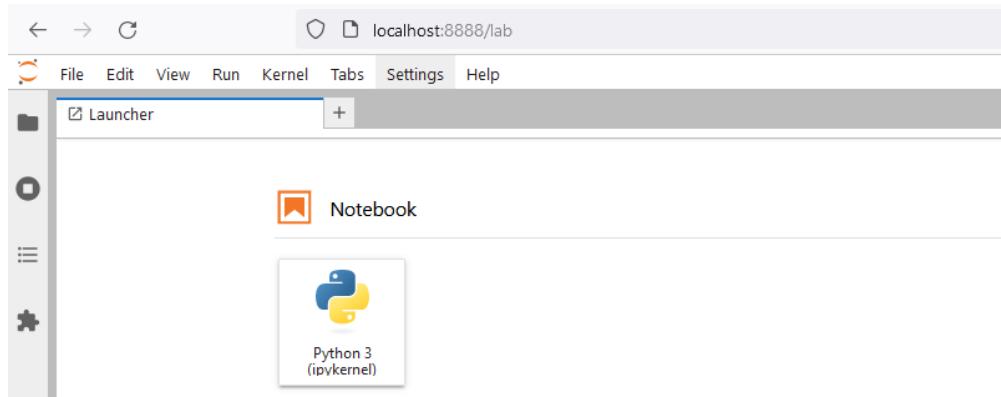
1.1. Creación del entorno con Jupyter

Si utilizaremos Jupyter Lab como editor para nuestro código, se deberán seguir los siguientes pasos, debe tener en cuenta que al utilizar un *notebook* de Jupyter la extensión del archivo creado será `.ipynb`, si deseamos guardarla como un *script* de Python, se deberá exportar al formato `.py`.

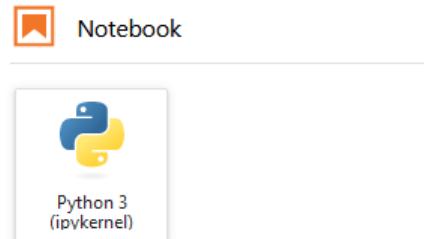
1. Debemos situarnos en la carpeta donde deseamos almacenar nuestro código y abrir una consola que apunte a dicha ubicación.
2. Esto se puede realizar llamando a la consola y haciendo los `cd` (cambios de directorios) respectivos o presionando *Shift + Click Derecho* en la carpeta deseada y seleccionando la opción *Abrir ventana de comandos aquí*.
3. Luego para activar el Jupyter Lab escribimos el siguiente código en la consola.

```
jupyter-lab
```

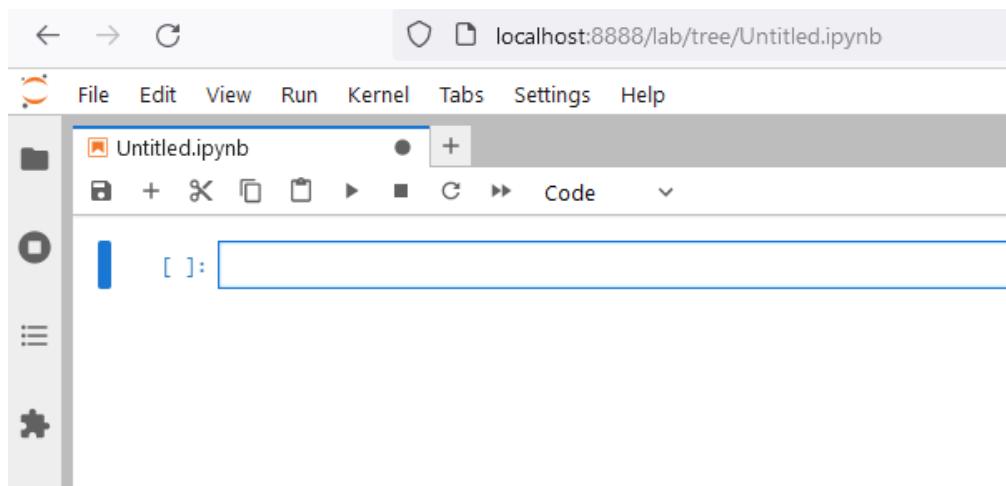
4. Lo cual luego de un momento abrirá una pestaña en nuestro navegador predeterminado de la siguiente manera:



5. Luego el siguiente paso consiste en crear nuestro *notebook* donde se escribirá el código y se ejecutará el mismo, para ello seleccionamos la opción Python 3 (o la ultima disponible en su computadora).



6. Esto abrirá una nueva pestaña que lucirá de la siguiente manera:



7. El espacio enmarcado en verde recibe el nombre de celda y es en este donde espacio donde se procederá a escribir código. Las celdas de este *notebook* son unidades de ejecución individual. Es decir cuando se ejecute una celda, solo se ejecutará el código que esta dentro de la misma. Un archivo puede tener una gran cantidad de celdas. Estas permiten ejecutar un código paso a paso para evaluar por partes los resultados que se van obteniendo.
8. Si en su caso desea trabajar desde la consola, la ejecución de un script de Python se realiza de la siguiente manera.

```
python nombreScript.py
```

9. De igual manera si utiliza VSCode, puede crear un archivo .ipynb y el IDE le proporcionará la interfaz con las celdas para colocar el código.

1.2. Trabajo inicial con imágenes

En esta sección se mostrará como leer una imagen desde archivo, como cambiar los espacios de color de una imagen y mostrar las imágenes que ha sido cargadas por OpenCV.

1. En nuestra primera celda copiamos la siguiente línea de importación, esta importará OpenCV en nuestro código.

```
import cv2
```

2. En una nueva celda copiamos el siguiente código:

```
# leer imagen
im = cv2.imread('img/tower.jpg')
# Dimensiones de la Imagen
h,w,c = im.shape
print("Dimensiones de la imagen - Alto: {},  

      Ancho: {}, Canales: {}".format(h, w, c))

# Numero total de Píxeles
tot_pix = im.size
print("Número total pixeles: {}".format(tot_pix))

# Obtener el tipo de datos de la imagen
image_dtype = im.dtype

# Imprimir tipo de datos de la imagen:
print("Tipo de datos de la imagen: {}".format(image_dtype))
```

3. Este fragmento de código permite leer la imagen tower.jpg dentro de la carpeta img de nuestra computadora.
4. El siguiente código obtiene las dimensiones de la imagen (alto, ancho, profundidad), el numero de píxeles y el tipo de dato, para después imprimir cada dato, con su respectiva leyenda.
5. Si deseamos convertir nuestra imagen a escala de grises podemos escribir lo siguiente:

```
#Crear una versión en escala de grises
gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
```

6. Para mostrar por pantalla la imagen cargada se utilizará la función cv2.imshow, la cual recibe como parámetros un título para la ventana y la imagen a mostrar.

```
# Mostrar imagen Original
cv2.imshow("Imagen Orginal", im)

# Mostrar Imagen en Escalas de Grises
cv2.imshow("Imagen en Escala de Grises", gray)

# Establecer espera
cv2.waitKey(0)
# Para destruir todas las ventanas creadas
cv2.destroyAllWindows()
```

7. La función `cv2.waitKey()` establece un tiempo de espera para un evento del teclado. Esta función se utiliza para mantener el visor de la imagen en pantalla. Recibe el tiempo de espera en milisegundos. Si una tecla se presiona en el tiempo definido, el programa continua su ejecución. Cuando recibe 0 como parámetro el sistema espera hasta que se presione una tecla.
8. La función `cv2.destroyAllWindows()` destruye todas las ventanas o visores que se han creado. Para una correcta ejecución de su programa siempre agregue estas dos funciones cuando trabaje con ventanas.
9. El utilizar este visor muestra la imagen a tamaño real, por lo cual en ocasiones puede ser más grande que la pantalla, por lo cual una estrategia es escalar la imagen para que se pueda ver en la pantalla. Para esto se agrega el siguiente código:

```
# Nuevo Tamaño
dsize = (int(w*0.250), int(h*0.250))

# escalar imagen
gray_rz = cv2.resize(gray, dsize)
im_rz = cv2.resize(im, dsize)
cv2.imshow("Imagen en Escala de Grises Escalada", gray_rz )
cv2.imshow("Imagen Orginal Escalada", im_rz)
cv2.waitKey(0)
# Para destruir todas las ventanas creadas
cv2.destroyAllWindows()
```

10. En esta definimos una nueva dimensión con la función `cv2.resize`, la cual recibe la imagen y las nuevas dimensiones y crea una nueva versión de la imagen pero con el tamaño definido.
11. Si deseamos almacenar las imágenes modificadas, utilizamos el siguiente código;

```
# guardar imagen
cv2.imwrite("out/gray.png",gray)
cv2.imwrite("out/tower2.png",im_rz)
```

12. Esta función recibe la ruta donde se guardará la nueva imagen y también el objeto imagen que fue modificado y se desea guardar.

1.3. Obtener información de un píxel

1. OpenCV permite acceder a los valores puntuales de color de un píxel utilizando sus coordenadas, esto retornará una arreglo con los valores (Blue, Green, Red).
2. Utilizando el código de la sección anterior podemos obtener los valores de un píxel para la imagen en memoria, de la siguiente manera:

```
# Obtener colores de un píxel
(b, g, r) = im[6, 40]

# Imprimir los valores:
print("Pixel (6,40) - Rojo: {}, Verde: {}, Azul: {}".format(r, g, b))
```

3. Se puede también acceder de manera individual a un canal

- Canal Azul (B): valor → 0
- Canal Verde (G): valor → 1
- Canal Rojo (R): valor → 2

```
# obtener información de un canal
b = im[6, 40, 0]

print("Pixel (6,40) - Azul: {}".format(b))
```

4. De la misma manera en la cual obtenemos la información de color de un píxel específico, es posible modificar el color de un punto en la imagen usando el siguiente código, se debe usar el formato (b, g, r) para indicar el nuevo color.

```
# Colocar un píxel en rojo
im_rz[6, 40] = (0, 0, 255)

# Obtener el valor del píxel (x=40, y=6) después de su modificación
(b, g, r) = im_rz[6, 40]

print("Pixel (6,40) - Rojo: {}, Verde: {}, Azul: {}".format(r, g, b))

cv2.imshow("Modificación de Píxel", im_rz)
cv2.waitKey(0)
# Para destruir todas las ventanas creadas
cv2.destroyAllWindows()
```

5. Es posible también seleccionar una sección o parche de la imagen en este caso se definen el rango de píxeles a tomar tanto en \hat{x} y \hat{y} , lo cual se puede hacer de la siguiente manera:

```
# En este ejemplo utilizamos la esquina sup. izq de la imagen

roi_izq_sup = im_rz[0:50, 0:50]

# Mostramos esta región de interés (ROI):
cv2.imshow("Esquina Superior Izquierda Original", roi_izq_sup)
cv2.waitKey(0)
```

```
# Se copia este parche a otra zona de la imagen  
im_rz[20:70, 20:70] = roi_izq_sup  
  
# Mostrar imagen modificada  
cv2.imshow("Imagen modificada", im_rz)  
cv2.waitKey(0)
```

-
6. Utilizando el concepto de parche, podemos cambiar el color de todos los píxeles dentro de un área de la imagen, como se ve en el siguiente código

```
# Cambiar el color de una zona a azul  
im_rz[0:50, 0:50] = (255, 0, 0)  
  
# Mostrar imagen modificada  
cv2.imshow("Imagen Modificada", im_rz)  
cv2.waitKey(0)  
  
cv2.destroyAllWindows()
```

Entregables para la Sección

Para esta sección en su informe debe incluir la modificación de una imagen de su propiedad. En dicha imagen se deben realizar las siguientes operaciones:

1. 1 modificación de color mediante la técnica del parche, el área modificar debe tener un forma rectangular y ubicarse en el centro de la imagen.
2. 1 modificación de color mediante la técnica del parche de forma cuadrada ubicada en la esquina inferior izquierda con el siguiente color RGB (100, 255, 51).
3. Crear un parche de una zona rectangular de la imagen y ubicarlo en la esquina inferior derecha de la imagen.
4. Todas las modificaciones se deben ver en una sola imagen.

2. Mostrar Imágenes con matplotlib

2.1. Espacio de Color en MatPlotLib

Matplotlib es una librería de Python que permite crear visualizaciones estáticas, animadas e interactivas. Esta librería permite la creación de gráficos tanto en *2D* como en *3D*. Funciona también como una herramienta para mostrar imágenes y los resultados obtenidos después de modificar imágenes con OpenCV.

Los canales de color que utiliza Matplotlib para pintar sus imágenes, están almacenados de manera diferente a la manera en la cual OpenCV almacena sus imágenes, por lo cual, cuando se desee mostrar una imagen almacenada en OpenCV, de manera directa en Matplotlib sus canales estarán en otro orden lo cual dará lugar a una imagen mal coloreada.

En esta sección veremos como trabajar con este aspecto para conseguir una visualización adecuada usando esta librería.

1. Como primer paso debemos crear un nuevo script/notebook dependiendo de su entorno de programación. Dicho archivo funcionará para todo el código de esta sección del laboratorio.
2. Dentro de este archivo agregamos el código de las importaciones. Se agrega Matplotlib y NumPy ya que será utilizado más adelante. En este caso ambas librerías se importan con su respectivo alias, el cual puede ser elegido a gusto del programador.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

3. El siguiente paso será cargar una imagen y obtener los canales que la componen mediante las siguientes líneas, específicamente con la función `cv2.split()`.

```
# Cargar una imagen
img_OpenCV = cv2.imread('img/tower2.png')

# Dividir la imagen en sus 3 canales (b, g, r):
b, g, r = cv2.split(img_OpenCV)

h,w,c = img_OpenCV.shape
print("Dimensiones de la imagen - Alto: {},\nAncho: {}, Canales: {}".format(h, w, c))
```

4. Debido al distinto orden de los canales de color para ver adecuadamente una imagen en Matplotlib debemos cambiar el orden en el cual se guardan dentro de la imagen, lo cual se realiza con la función `cv2.merge()`:

```
# Combinar los canales pero en orden RGB
img_matplotlib = cv2.merge([r, g, b])
```

5. En este punto contamos con 2 copias de la misma imagen, pero con sus canales almacenados en un orden diferente, para verlas podemos utilizar la función `subplot()` de MatPlotLib.

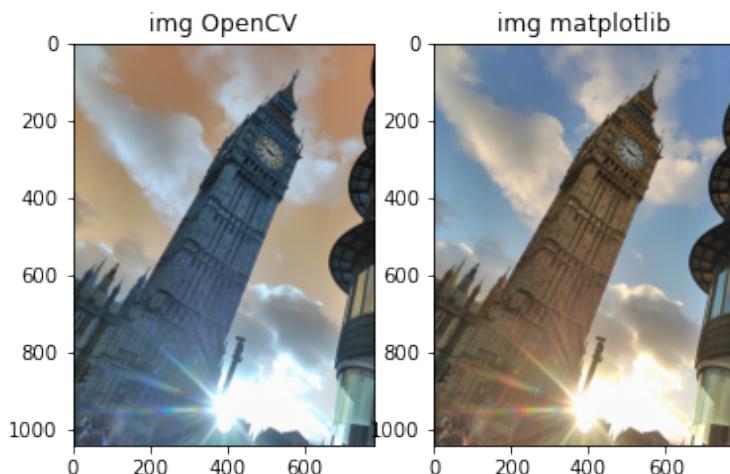
La función `subplot()` permite crear una visualización compuesta de varias imágenes las cuales podemos ordenar haciendo referencia a las coordenadas del *plot*. Esta función recibe una triada de valores que indican la configuración del *plot* y la ubicación donde deseamos colocar nuestra imagen. Por lo cual, estos 3 números se interpretan como (*NumFilas, NumColumnas, Indice*). De tal manera que una instrucción como `plt.subplot(2,2,1)` indica que se crea matriz o grid de tamaño 2×2 y que la gráfica deseada se colocará en el posición 1 dentro de esa matriz, las posiciones se enumeran de izquierda a derecha y de arriba hacia abajo.

Esta función también puede recibir un número de 3 dígitos de la siguiente manera `plt.subplot(221)` en este caso se separará dicho número y se interpretará igual que en el caso anterior. Para más información sobre esta función y su forma de operar puede consultar este [enlace](#).

6. Por consiguiente para apreciar nuestras imágenes utilizando MatPlotLib debemos escribir el siguiente código:

```
# Ver ambas imágenes(img_OpenCV y img_matplotlib) usando matplotlib
# Esta mostrará la imagen con los colores erróneos
plt.subplot(121)
plt.imshow(img_OpenCV)
plt.title('img OpenCV')
# Esto mostrará los colores verdaderos
plt.subplot(122)
plt.imshow(img_matplotlib)
plt.title('img matplotlib')
plt.show()
```

7. Lo cual generaría el siguiente resultado



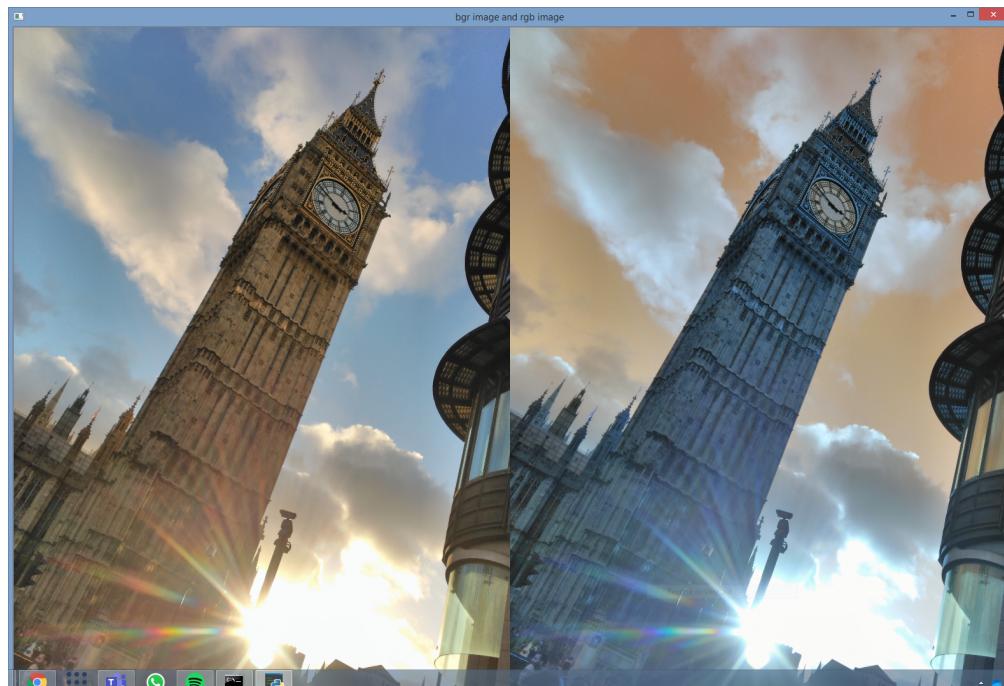
8. Si deseamos ver las imágenes utilizando OpenCV, podemos copiar lo siguiente:

```
# Ver ambas imágenes(img_OpenCV y img_matplotlib) usando cv2.imshow()  
# Esto mostrará los colores verdaderos  
cv2.imshow('bgr image', img_OpenCV)  
# Esta mostrará la imagen con los colores erróneos  
cv2.imshow('rgb image', img_matplotlib)  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

9. Esto nos produce una visualización en ventanas separadas, pero si se usa la función np.concatenate de NumPy, podemos unir ambas imágenes y mostrarlas en una sola ventana. En este caso la altura de ambas imágenes debe ser similar. El parámetro axis indica en que dirección se concatenaran las imágenes (horizontal en este caso).

```
# Concatenar Imagenes Horizontalmente  
# (img_OpenCV a la izquierda de img_matplotlib):  
img_concats = np.concatenate((img_OpenCV, img_matplotlib), axis=1)  
  
# Mostrar la imagen concatenada  
cv2.imshow('bgr image and rgb image', img_concats)  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

10. Lo cual generaría el siguiente resultado



2.2. Utilizando NumPy con estructuras de imágenes

En esta sección se seguirá utilizando el código creado para la sección anterior (Matplotlib), por lo cual, solo deberá agregar los nuevos fragmentos al código anterior.

NumPy es una librería de Python para trabajar con arreglos N-Dimensionales, en Python las imágenes son almacenadas en una estructura de tipo matriz usando NumPy, por lo cual se pueden modificar utilizando los mecanismos de esta librería.

1. Como primer paso con NumPy podemos obtener los canales de la imagen y construir una nueva, similar a la forma de trabajo de cv2.split(), esto se logra con el siguiente fragmento:

```
# Obtener los canales, similar a cv2.split()
B = img_OpenCV[:, :, 0]
G = img_OpenCV[:, :, 1]
R = img_OpenCV[:, :, 2]
```

2. De igual manera es posible realizar la conversión de colores con NumPy mediante un iterable que extrae los canales en orden inverso, lo cual se logra con el siguiente código. En este caso se modifica la imagen en formato OpenCV y se muestra con el visor de OpenCV, por lo cual los colores se apreciarán erróneamente.

```
# Transformar la imagen BGR a RGB usando Numpy :
# ::-1 iterable que imprime los elementos en orden inverso
img_RGB = img_OpenCV[:, :, ::-1]

# Ahora se muestra la imagen, pero los colores serán erróneos
cv2.imshow('img RGB (wrong color)', img_RGB)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. Lo cual generaría el siguiente resultado



4. Con NumPy podemos también crear imágenes utilizando solamente código fuente como se muestra en el siguiente fragmento, en esta imagen el valor de cada píxel será 0.

```
# Creación de una imagen con dimensiones 1040x580 y 3 canales de color
bgr = np.zeros((1040, 580, 3), dtype=np.uint8)
```

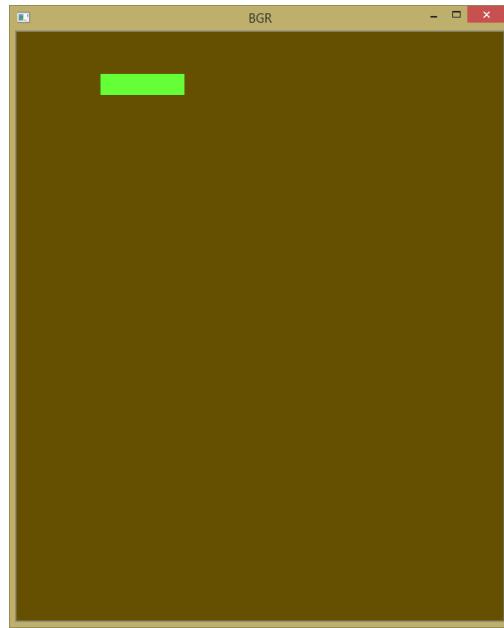
5. Para modificar los colores asignados a los píxeles podemos utilizar los siguientes códigos, dependiendo de las zonas que deseemos pintar.

```
# Asignar un color a todos los píxeles de la imagen
bgr[:, :, :] = (0, 80, 100)

# Asignar un color a una sección de la imagen
bgr[50:75,100:200,:] = (55, 255, 100)

# Mostrar la imagen creada
cv2.imshow('BGR', bgr)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

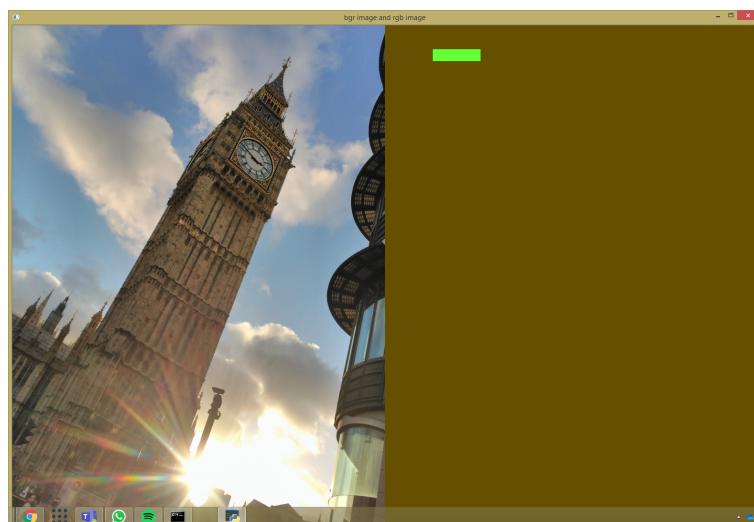
6. Lo cual generaría el siguiente resultado



7. De igual manera una imagen creada con NumPy se puede concatenar con otra imagen usando la función `np.concatenate`, como el siguiente código:

```
# Concatenar Horizontalmente 2 imágenes
img_concat = np.concatenate((img_OpenCV, bgr), axis=1)
cv2.imshow('bgr image and rgb image', img_concat)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8. Lo cual generaría el siguiente resultado



Entregables para la Sección

Para esta sección en su informe debe incluir la modificación de una imagen de su propiedad. Con dicha imagen se deben realizar las siguientes operaciones:

1. Leer un imagen de su computadora se recomienda que esta imagen sea en formato horizontal o landscape. Convertir la imagen a formato de Matplotlib (RGB).
2. Crear una imagen utilizando NumPy y asignar al fondo el color RGB (51, 184, 255), crear un recuadro en dicha imagen que debe ser del siguiente color RGB (255, 51, 200), esta imagen debe ser en formato horizontal o apaisada.
3. Concatene verticalmente las 3 imágenes (RGB, NumPy, BGR) utilizando la función `np.concatenate()` y adjunte la imagen generada a su informe.
4. Concatene verticalmente las 3 imágenes (RGB, NumPy, BGR) utilizando la función `subplot` y adjunte la imagen o una captura de esta generada a su informe.
5. Todas las modificaciones se deben ver en una sola imagen.

3. Paso de argumentos y Utilización de una Cámara

En esta sección se muestra un código completo y funcional que permite utilizar una cámara conectada al computador para hacer una fotografía y guardarla en nuestro sistema. Se utiliza también la funcionalidad de Python para enviar elementos o argumentos mediante la consola.

En este caso se utiliza la librería argparse para realizar un análisis de los elementos pasados a la consola al invocar la ejecución del programa.

Para este script, se debe indicar un numero entero que señala la cámara a utilizar, en el caso solo tener una cámara se debe enviar el número 0 de la siguiente manera.

```
python lab1_camara.py 0
```

En este ejemplo al presionar la letra *c* se almacena lo que este viendo la cámara en ese preciso instante. Presionando la letra *q* se termina la ejecución del programa.

El código completo se muestra a continuación:

```
"""
Ejemplo para obtener una imagen de la cámara y almacenarla en el equipo
"""

import cv2
import argparse

# Se crea el objeto ArgumentParser
# Este objeto tendra la información necesaria para
# analizar los argumento de la linea de comando en sus tipos de datos.
parser = argparse.ArgumentParser()

# Se añade el argumento 'index_camera' usando el add_argument()
# e incluir la ayuda.
parser.add_argument("index_camera",
                    help="index of the camera to read from", type=int)
args = parser.parse_args()

# Definir la camara a utilizar
capture = cv2.VideoCapture(args.index_camera)

# Obtener ciertas propiedades del dispositivo de captura
#(frame width, frame height y frames per second (fps)):
frame_width = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
frame_height = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)
fps = capture.get(cv2.CAP_PROP_FPS)

# Imprimir estos valores
print("CV_CAP_PROP_FRAME_WIDTH: {}".format(frame_width))
print("CV_CAP_PROP_FRAME_HEIGHT : {}".format(frame_height))
print("CAP_PROP_FPS : {}".format(fps))
```

```
# Verificar si la camara se abrio adecuadamente
if capture.isOpened() is False:
    print("Error al abrir la camara")

# Contador para guardar las imágenes
frame_index = 0

# Leer hasta que se completen las acciones
while capture.isOpened():
    # Capturar frame-by-frame la información de la cámara
    ret, frame = capture.read()

    if ret is True:
        # Mostrar el frame capturado
        cv2.imshow('Entrada de la camara', frame)

        # Convertir el frame a escala de grises
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Mostrar el frame en escala de grises:
        cv2.imshow('Entrada en Escala de Grises', gray_frame)

        # Presionar C para guardar
        if cv2.waitKey(20) & 0xFF == ord('c'):
            frame_name = "out/camera_frame_{}.png".format(frame_index)
            gray_frame_name = "out/grayscale_camera_frame_{}.png".format(frame_index)
            cv2.imwrite(frame_name, frame)
            cv2.imwrite(gray_frame_name, gray_frame)
            frame_index += 1

        # Presionar q para salir
        if cv2.waitKey(20) & 0xFF == ord('q'):
            break
    # Romper el bucle
else:
    break

# Liberar camara y limpiar ventanas
capture.release()
cv2.destroyAllWindows()
```

Entregables para la Sección

Para esta sección en no se debe entregar ninguna evidencia en su informe de laboratorio.