Nicolalde Willams - Tarea 3 - DocCompleto

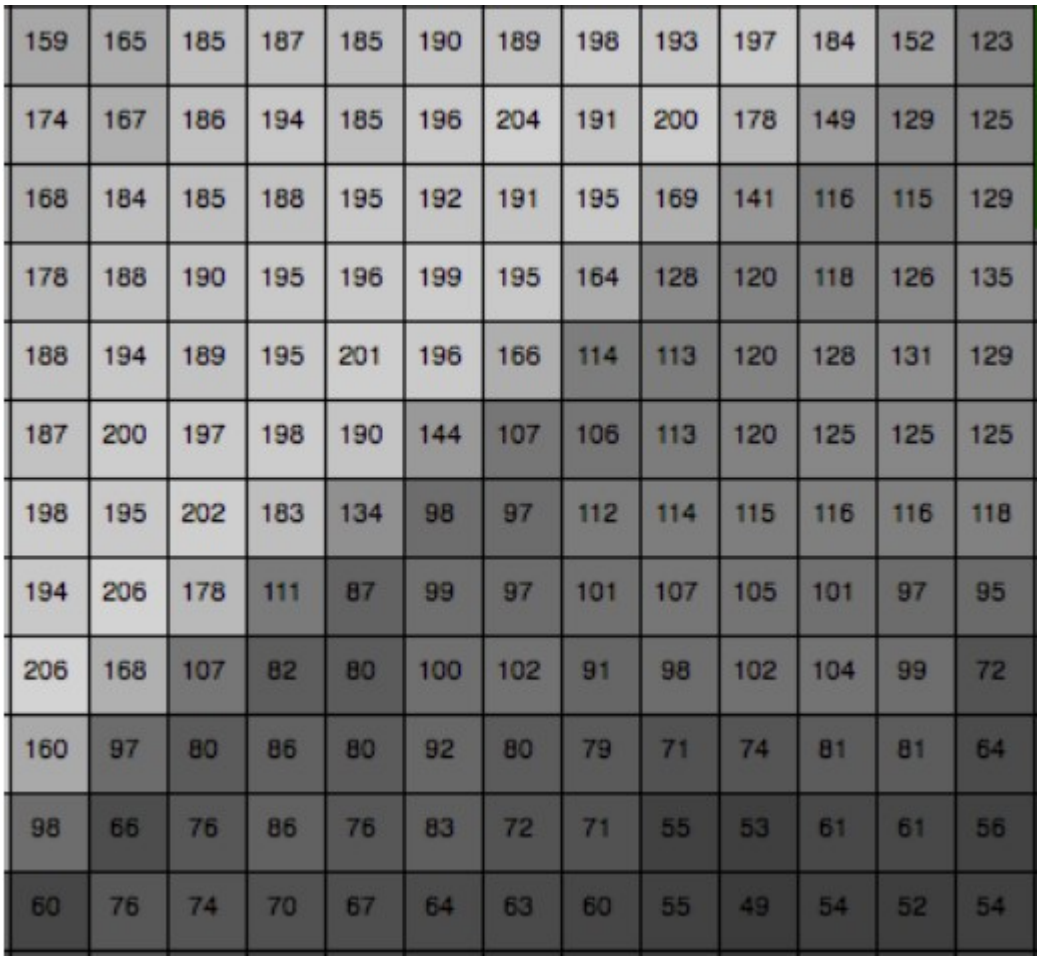
November 19, 2020

0.1 1 Imagénes y Matrices

La estructura más importante en una visión artificial es, sin duda, las imágenes. La imagen en visión artificial es una representación del mundo físico capturado con un dispositivo digital. Esta imagen es solo una secuencia de números almacenados en un formato de matriz, como se muestra en la siguiente imagen. Cada número es una medida de la intensidad de la luz para la longitud de onda considerada (por ejemplo, en rojo, verde o azul en imágenes en color) o para un rango de longitud de onda (para dispositivos pancromáticos). Cada punto de una imagen se llama píxel (para un elemento de imagen), y cada píxel puede almacenar uno o más valores dependiendo de si es una imagen gris, negra o blanca (también llamada imagen binaria) que almacena solo uno valor, como 0 o 1, una imagen de nivel de escala de grises que puede almacenar solo un valor, o una imagen en color que puede almacenar tres valores. Estos valores suelen ser números enteros entre 0 7y 255, pero puede usar el otro rango. Por ejemplo, 0 a 1 en números de coma flotante como HDRI (imágenes de alto rango dinámico) o imágenes térmicas.

La imagen es almacenado en un formato matricial, donde cada pixel tiene una posicion y puede ser referenciado por su fila y columna. En el caso de una imagen en escala de grises una matris simple

representa la imagen.



En el caso de una imagen a color, como mostramos en la figura, sera represntada por una matriz de ancho * alto * numero de colores (RGB)

0.2 Introducción a OpenCV

0.2.1 2.1 Leer, Escribir y Mostrar una imagen

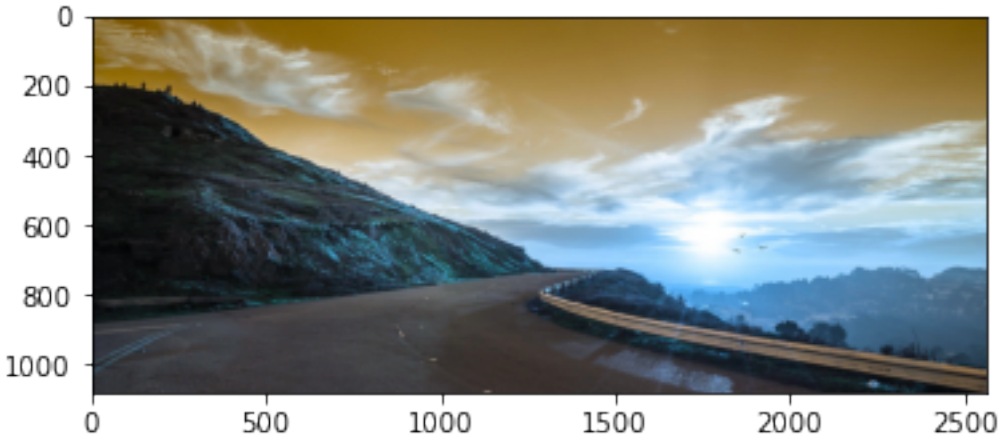
OpenCV provee las funciones imread() y imwrite() que soportan varios formatos de archivos para trabajar con imagenes, la imagen debe estar en el directorio de trabaja actual o debes pasar la direccion absoluta o relativa de la ubicacion de la imagen.

2.1.1 imread()

[2]:
```
# importar la libreria de CV
import cv2 as cv
import matplotlib.pyplot as plt

# lectura de una imagen
img = cv.imread("./ImgT1/Horizon.jpg")
```

```
# mostrar la imagen
plt.imshow(img)
```

[2]: <matplotlib.image.AxesImage at 0x254b1e32a60>



Por defecto imread() retorna una imagen en formato de color BGR, incluso si usamos una imagen en formato de escala de grises. BGR representa el mismo espacio de color que RGB pero el orden de byte esta invertido.

2.1.2 Modos imread() (Banderas) IMREAD_UNCHANGED Python:

cv.IMREAD_UNCHANGED Si se establece, devuelve la imagen cargada como está (con canal alfa; de lo contrario, se recorta). Ignore la orientación EXIF.

IMREAD_GRAYSCALE Python: cv.IMREAD_GRAYSCALE Si está configurado, siempre convierta la imagen a la imagen en escala de grises de un solo canal (conversión interna del códec).

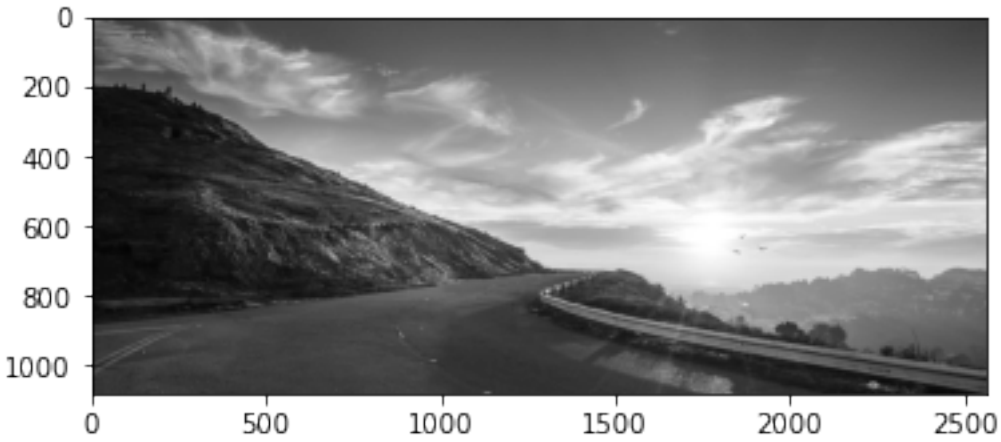
IMREAD_COLOR Python: cv.IMREAD_COLOR Si está configurado, siempre convierta la imagen a la imagen en color BGR de 3 canales.

IMREAD_ANYCOLOR Python: cv.IMREAD_ANYCOLOR Si se establece, la imagen se lee en cualquier formato de color posible.

2.1.3 cv.IMREAD_GRAYSCALE

[3]:
```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("./ImgT1/Horizon.jpg", cv.IMREAD_GRAYSCALE)
plt.imshow(img, cmap="gray")
```

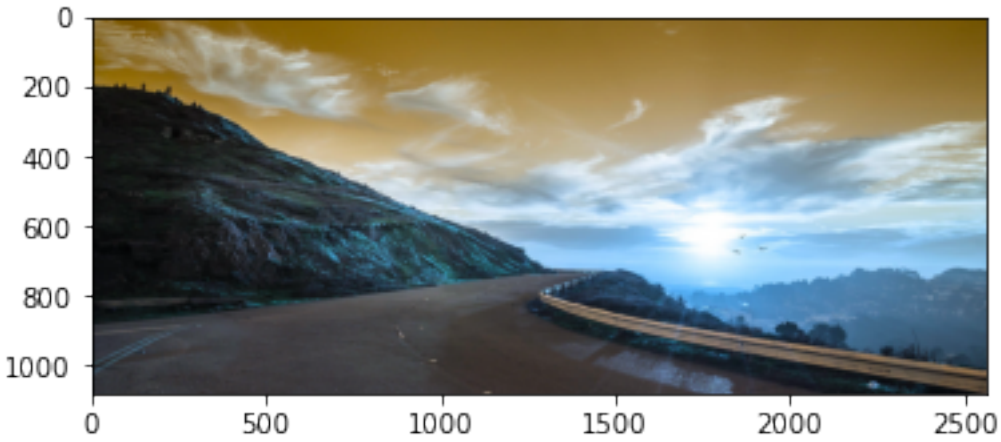
[3]: <matplotlib.image.AxesImage at 0x254b1ea65e0>



2.1.4 cv.IMREAD_COLOR

[4]:
```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("./ImgT1/Horizon.jpg", cv.IMREAD_COLOR)
plt.imshow(img, cmap="gray")
```

[4]: <matplotlib.image.AxesImage at 0x254b216ad90>



[5]:
```
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

[5]: <matplotlib.image.AxesImage at 0x254b21c92b0>

2.1.5 imshow()

[6]:
```
import cv2 as cv
# Leemos la imagen
img = cv.imread("./ImgT1/Horizon.jpg")
cv.imshow("Horizon", img)
# Cerramos la imagen
cv.waitKey(0)
cv.destroyAllWindows()
```

2.1.6 imwrite() La funcion imwrite nos permite escribir en disco o guardar una imagen, le pasamos como argumento la ruta donde donde guardar con el nombre a guardar y la extencion, y como segundo argumento la imagen a guardar si todo sale sin errores devuelve True.

[7]:
```
import cv2 as cv
img = cv.imread("./ImgT1/Horizon.jpg")
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imwrite("./ImgT1/Horizon_gray.jpg", img_gray)
```

[7]:
```
True
```

0.2.2 2.2 Lectura y Escritura de Archivos de video

OpenCV dispone de las clases VideoCapture() y VideoWriter() que soporta varios formatos de archivos de video Los formatos admitidos varían según el sistema, pero siempre deben incluir AVI. A través de su método read (), una clase de VideoCapture puede sondearse para nuevos cuadros hasta llegar al final de su archivo de video. Cada cuadro es una imagen en formato BGR.

Reproducir video desde un archivo

[10]:
```
import numpy as np
import cv2 as cv
```

```
# crear un objeto VideoCapture su parametro sera la direccion del video
cap = cv.VideoCapture("../Videos/NimbusApp.mp4")
# creamos un bucle
while(cap.isOpened()):

   ret,frame = cap.read()
   if ret:

       cv.imshow("Aplicación NimbusAds", frame)

       if(cv.waitKey(10) & 0xFF ==ord("q")):
           break
   else:
       break
cap.release()
cv.destroyAllWindows()
```

Acceder a la cámara

[11]:
```
import numpy as np
import cv2 as cv
# crear un objeto VideoCapture su parametro sera la direccion del video (0 para␣
 ↪webcam)
cap = cv.VideoCapture(0)
# creamos un bucle
while(cap.isOpened()):

   ret,frame = cap.read()
   if ret:

       cv.imshow("Camara Activa", frame)

       if(cv.waitKey(10) & 0xFF ==ord("q")):
           break
   else:
       break
cap.release()
cv.destroyAllWindows()
```

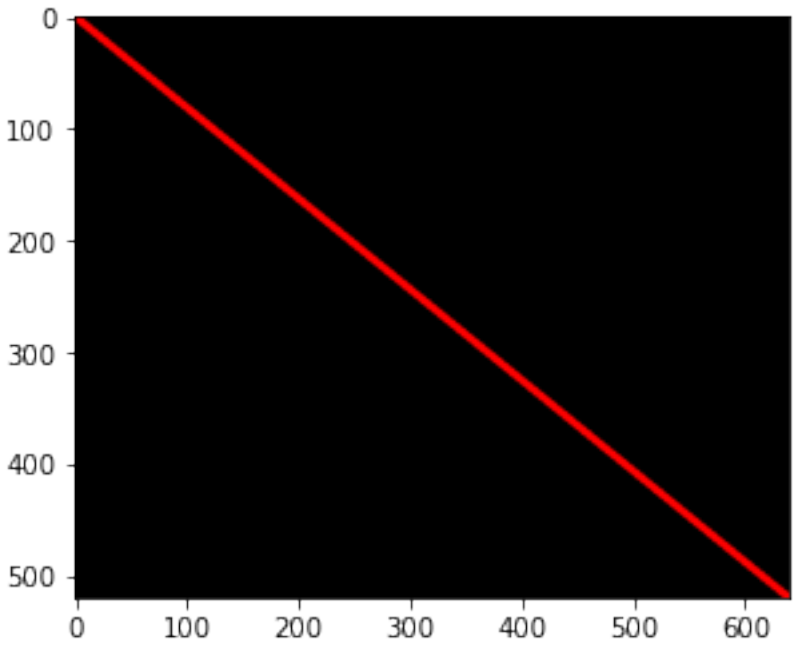
0.3 Funciones para dibujar figuras geométricas

0.3.1 Línea

[12]:
```
#función cv.line(imagen, coordenadas iniciales,
#coordenadas finales, color, grosor)
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

```
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar la linea
img = cv.line(img,(0,0),(640,520),(255,0,0),6)
#mostrar imagen
plt.imshow(img)
```

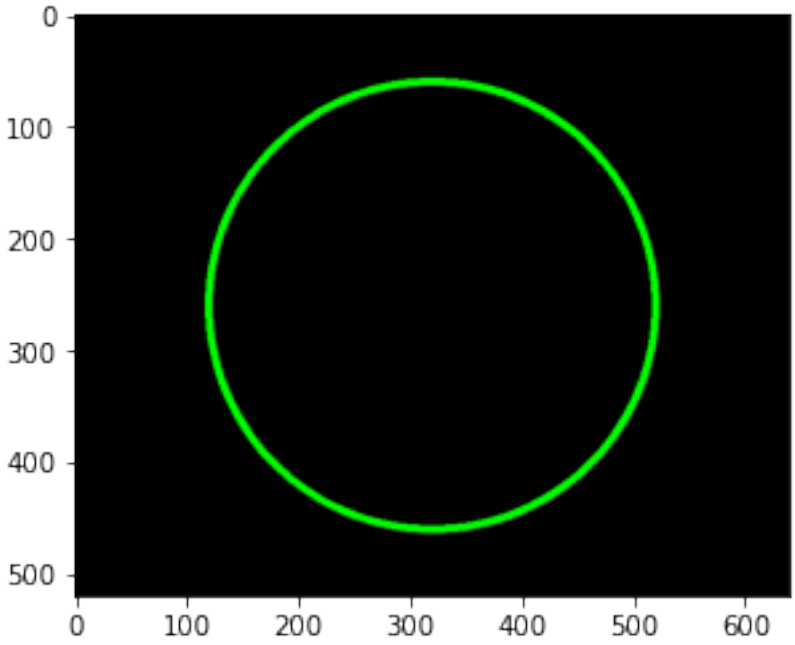
[12]: <matplotlib.image.AxesImage at 0x254b2dcc1c0>



0.3.2 Dibujar un círculo

[13]:
```
#función cv.circle()
#parámetros: img, coordenadas centro, radio, color, grosor
#función cv.line(imagen, coordenadas iniciales,
#coordenadas finales, color, grosor)
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar círculo
img = cv.circle(img,(320,260),200,(0,255,0),5)
#mostrar imagen
plt.imshow(img)
```

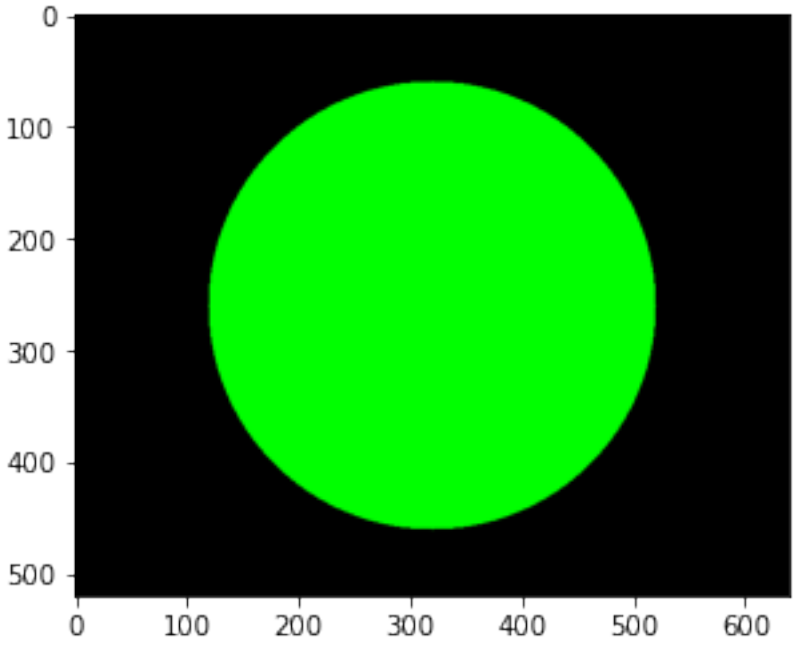
[13]: <matplotlib.image.AxesImage at 0x254b3be8d00>



0.3.3 Círculo relleno

[14]:
```
#función cv.circle()
#parámetros: img, coordenadas centro, radio, color, grosor
#función cv.line(imagen, coordenadas iniciales,
#coordenadas finales, color, grosor)
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar círculo (-1 para rellenar el círculo)
img = cv.circle(img,(320,260),200,(0,255,0),-1)
#mostrar imagen
plt.imshow(img)
```

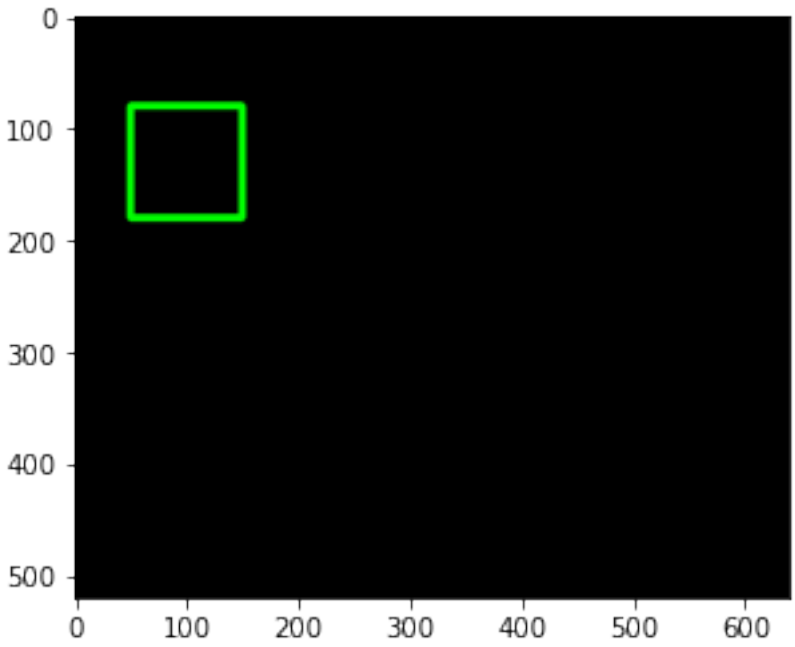
[14]: <matplotlib.image.AxesImage at 0x254b3c449a0>



0.3.4 Dibujar rectángulo

[15]:
```
#función cv.rectangle()
#parámetros: img, coordenadas esquina superior izq, coordenada esquina inf der,␣
↪color, grosor
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar rectangulo
img = cv.rectangle(img,(50,80),(150,180),(0,255,0),5)
#mostrar imagen
plt.imshow(img)
```

[15]:
```
<matplotlib.image.AxesImage at 0x254b3ca0670>
```



0.4 Captura de eventos del Mouse

OpenCV permite que las ventanas con nombre se creen, redibujen y destruyan usando las funciones namedWindow(), imshow() y destroyWindow(). Además, cualquier ventana puede capturar la entrada del teclado a través de la función waitKey () y la entrada del mouse a través de la función setMouseCallback(). Veamos un ejemplo donde mostramos cuadros de entrada de cámara en vivo

0.4.1 Función setMouseCallback()

[16]:
```
import cv2 as cv
clicked = False
#funcion encargada de capturar el mouse
def onMouse(evento,x,y,flags,param):
    #variable glogal
    global clicked

    if(evento == cv.EVENT_LBUTTONUP):
        clicked = True
#crear objeto video cv.VideoCapture()
camCap = cv.VideoCapture(0)
cv.namedWindow("MyWindow")
#configurar envío de eventos a función onMouse
cv.setMouseCallback("MyWindow",onMouse)
print("Mostrar estado cámara, click en la ventana para detener")
ret,frame = camCap.read()
```

```
while(ret and cv.waitKey(1)==-1 and not clicked):
   cv.imshow("MyWindow", frame)
   ret, frame = camCap.read()
camCap.release()
cv.destroyAllWindows()
```

```
Mostrar estado cámara, click en la ventana para detener
```

El argumento para waitKey () es un número de milisegundos para esperar la entrada del teclado. El valor de retorno es -1 (lo que significa que no se ha presionado ninguna tecla) o un código de tecla ASCII, como 27 para Esc. Podemos enumerar todos los eventos del mouse disponibles con el siguiente codigo:

[17]:
```
import cv2 as cv
events = [i for i in dir(cv) if 'EVENT' in i ]
events
```

[17]:
```
['EVENT_FLAG_ALTKEY',
 'EVENT_FLAG_CTRLKEY',
 'EVENT_FLAG_LBUTTON',
 'EVENT_FLAG_MBUTTON',
 'EVENT_FLAG_RBUTTON',
 'EVENT_FLAG_SHIFTKEY',
 'EVENT_LBUTTONDBLCLK',
 'EVENT_LBUTTONDOWN',
 'EVENT_LBUTTONUP',
 'EVENT_MBUTTONDBLCLK',
 'EVENT_MBUTTONDOWN',
 'EVENT_MBUTTONUP',
 'EVENT_MOUSEHWHEEL',
 'EVENT_MOUSEMOVE',
 'EVENT_MOUSEWHEEL',
 'EVENT_RBUTTONDBLCLK',
 'EVENT_RBUTTONDOWN',
 'EVENT_RBUTTONUP']
```

0.4.2 Dibujar círculos en donde se haga click

[18]:
```
import numpy as np
import cv2 as cv
def draw_circle(event, x, y, flags, param):

   if(event == cv.EVENT_LBUTTONDBLCLK):

       cv.circle(img, (x, y), 25, (255,0,0), -1 )

img = np.zeros((512,512,3), np.uint8)
cv.namedWindow('Dibujando Circulos')
```

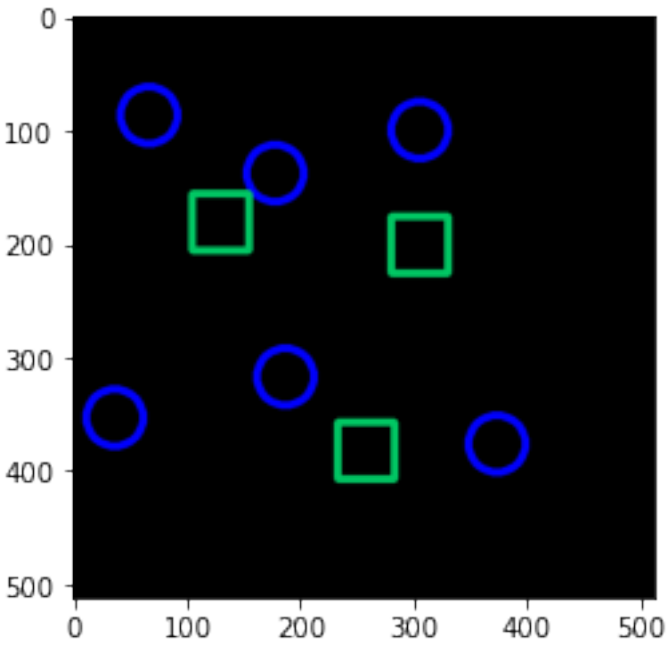
```
cv.setMouseCallback('Dibujando Circulos', draw_circle)
while(True):
   cv.imshow('Dibujando Circulos',img)
   if (cv.waitKey(1) & 0xFF ==ord('q')):
       break

cv.destroyAllWindows()
```

0.5 TAREA 1

Realizar un ejercicio capturando eventos con el mouse donde se pulse el Botón izquierdo dibuje un circulo, pulsamos el botón derecho dibuja un rectángulo.

[19]:
```
# Importar librerías
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

# Definir función para dibujar las figuras
def draw_figures(event, x, y, flags, param):

    # Dibujar circulos al dar click izquierdo
    if(event == cv.EVENT_LBUTTONDBLCLK):
        cv.circle(img, (x, y), 25, (255,0,0), 5 )

    # Dibujar rectangulos al dar click derecho
    if(event == cv.EVENT_RBUTTONDBLCLK):
        cv.rectangle(img, (x-25, y-25), (x+25, y+25), (100, 200, 0), 5)
# Crear la imagen
img = np.zeros((512, 512, 3), np.uint8)

cv.namedWindow('Tarea 1')
cv.setMouseCallback('Tarea 1', draw_figures)

while(True):
    cv.imshow('Tarea 1', img)
    if (cv.waitKey(1) & 0xFF ==ord('q')):
        break

cv.destroyAllWindows()
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

[19]:
```
<matplotlib.image.AxesImage at 0x254b3d07460>
```



0.6 3 Operaciones con Imágenes

Las operaciones con imágenes es la implementación de operaciones aritméticas estándar, como suma, resta, multiplicación y división, en imágenes. Estas operaciones tiene muchos usos en el procesamiento de imágenes, Por ejemplo,la resta de imagen se puede utilizar para detectar diferencias entre dos o más imágenes de la misma escena u objeto.

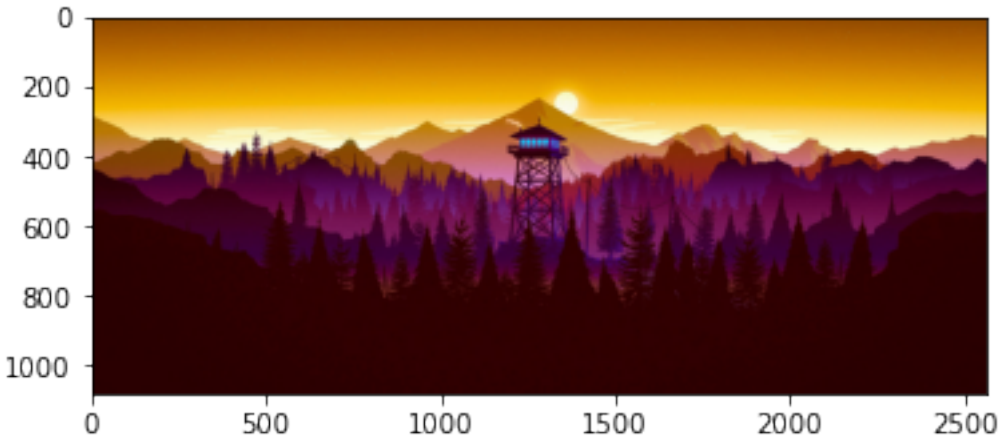
0.6.1 3.1 Operaciones Básicas con Imágenes

La mayoria de operaciones con imagenes estan relacionadas principalmente con la liberia numpy en lugar de OpenCV. Se requiere de conocimientos basicos en su uso para poder entender y aplicar de mejor manera las operaciones.

3.1.1 Acceso y Modificación de Pixeles

[21]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("ImgT2/Horizon Forest.jpg")
plt.imshow(img)
```

[21]:
```
<matplotlib.image.AxesImage at 0x254b3d84640>
```



Acceso Puede acceder a un valor de píxel por sus coordenadas de fila y columna. Para la imagen BGR, devuelve una matriz de valores azules, verdes y rojos. Para la imagen en escala de grises, solo se devuelve la intensidad correspondiente entre 0 y 255 siendo 0 igual a negro y 255 equivalente a blanco.

[22]:
```
# Accedemos al valor del pixel en sus 3 canales
px = img[100,100]
px
```

[22]:
```
array([182, 102,   1], dtype=uint8)
```

[23]:
```
# Accediendo al valor de un pixel en un solo canal canal B
px = img[100,100,0]
px
```

[23]:
```
182
```

[24]:
```
# Accediendo al valor de un pixel en un solo canal canal G
px = img[100,100,1]
px
```

[24]:
```
102
```

[25]:
```
# Accediendo al valor de un pixel en un solo canal canal R
px = img[100,100,2]
px
```

[25]:
```
1
```

3.1.2 Modificación

[26]:
```
# Modificando valor de pixeles
img[100,100]=[255,255,255]
```

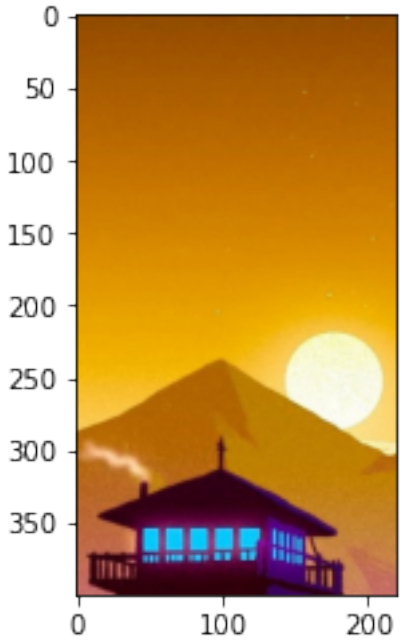
```
img[100,100]
```

[26]:
```
array([255, 255, 255], dtype=uint8)
```

Numpy es una biblioteca o libreria optimizada para calculos rapidos de matrices. No se recomienda simplemente acceder a cada valor de pixel de las formas ya vistas y modificarlos, porque seria un proceso muy lento. Estos metodos se usan para seleccionar una region de una matriz, supongamos las 5 primeras filas y las ultimas 3 columnas de la forma que la veremos a continuación

[27]:
```
img2 = img[:400, 1180:1400]
plt.imshow(img2)
```

[27]:
```
<matplotlib.image.AxesImage at 0x254b1e80850>
```



Metodos item() y itemset() Para el acceso individual de píxeles, los métodos de Numpy, array.item () y array.itemset () se consideran mejores. Sin embargo, siempre devuelven un escalar, por lo que si desea acceder a todos los valores B, G, R, deberá llamar a array.item () por separado para cada valor.

[28]:
```
# Accediendo
img.item(50,50,2)
```

[28]:
```
3
```

[29]:
```
# Modificando
img.itemset((50,50,2),100)
img.item(50,50,2)
```

[29]:
```
100
```

3.1.3 Acediendo a las Propiedades de la Imagen Las Imagenes poseen propiedades como numero de filas, numero de columnas, numero de pixeles, tipo de dato

3.1.4 Metodo shape

[30]:
```
img.shape
```

[30]:
```
(1080, 2560, 3)
```

3.1.4 Metodo dtype

[31]:
```
img.dtype
```

[31]:
```
dtype('uint8')
```

[32]:
```
type(img)
```

[32]:
```
numpy.ndarray
```

[33]:
```
img.size
```

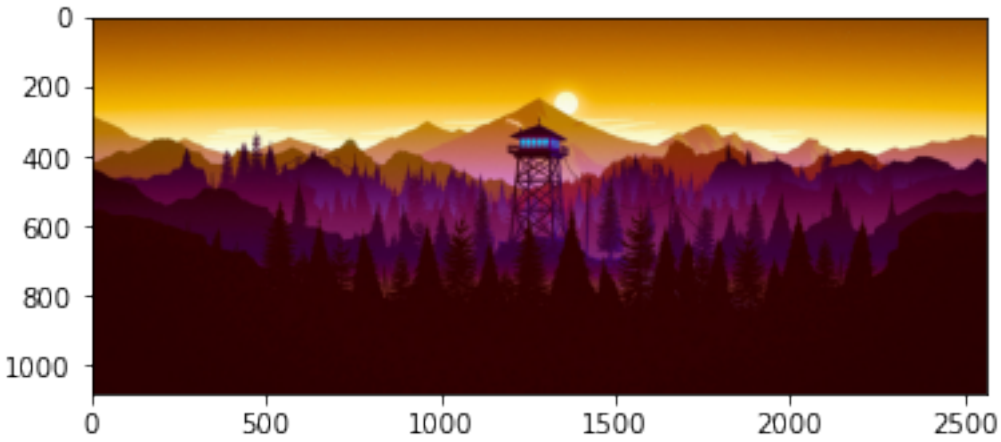
[33]:
```
8294400
```

0.6.2 3.2 ROI de una imagen

A veces, tendrás que jugar con ciertas regiones de imágenes. Para la detección de ojos en imágenes, la detección de la primera cara se realiza sobre toda la imagen. Cuando se obtiene una cara, seleccionamos solo la región de la cara y buscamos ojos dentro de ella en lugar de buscar en toda la imagen. Mejora la precisión (porque los ojos siempre están en las caras) y el rendimiento (porque buscamos en un área pequeña) El ROI lo podemos obtener utilizando la indexacion antes vista, en este ejemplo seleccionaremos la pelota y la copiaremos en otro lugar de la imagen:

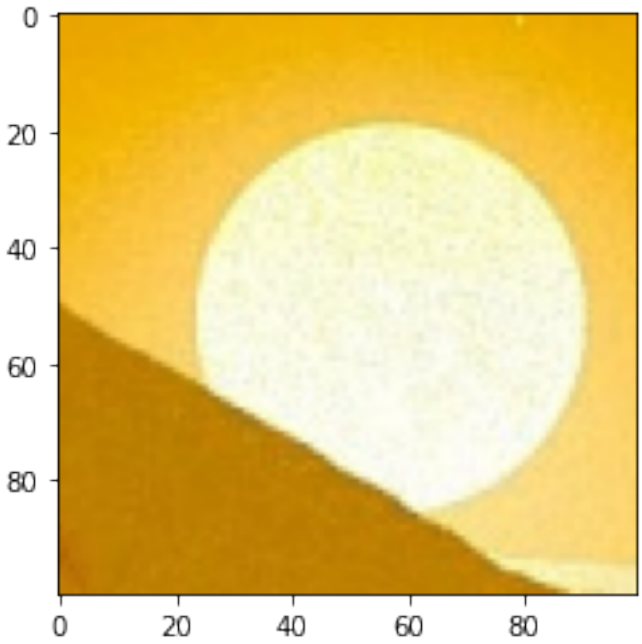
[34]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("ImgT2/Horizon Forest.jpg")
plt.imshow(img)
```

[34]:
```
<matplotlib.image.AxesImage at 0x254b3c10f70>
```



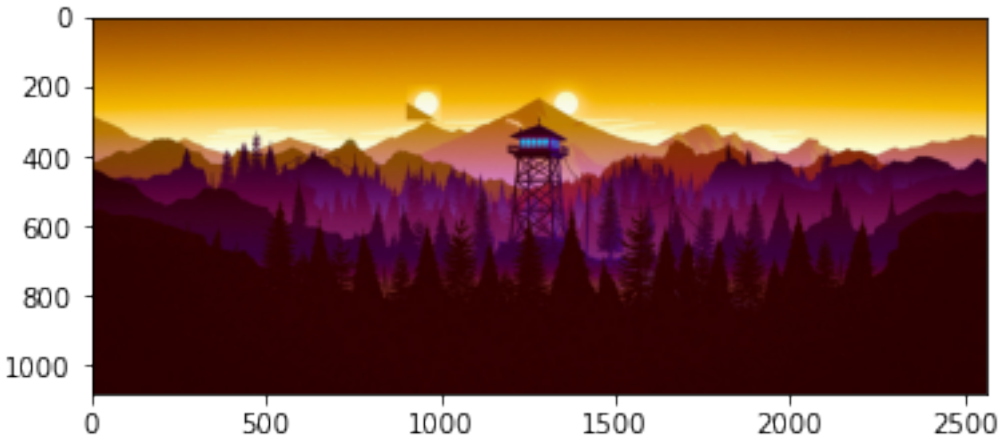
[35]:
```
# extraer la region de interes
sol = img[200:300,1300:1400]
plt.imshow(sol)
```

[35]: <matplotlib.image.AxesImage at 0x254b3e76790>



[36]:
```
img[200:300, 900:1000]=sol
plt.imshow(img)
```

[36]: <matplotlib.image.AxesImage at 0x254b3eca3d0>



0.6.3 División y Fusión de Canales

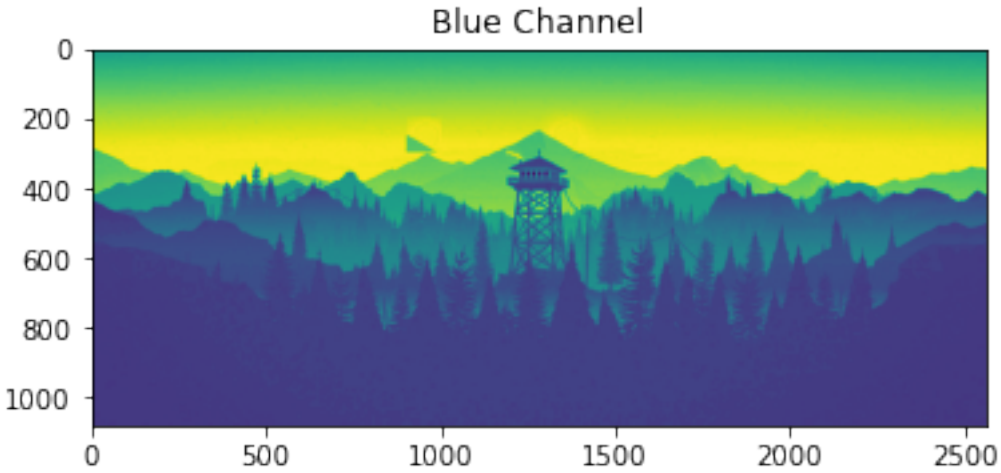
A veces necesitará trabajar por separado en los canales B, G, R de una imagen. En este caso, debe dividir la imagen BGR en canales individuales. En otros casos, es posible que deba unirse a estos canales individuales para crear una imagen BGR

3.3.1 split()

[37]:
```
# split() nos permite dividir los canales de una imagen
b,g,r = cv.split(img)
```

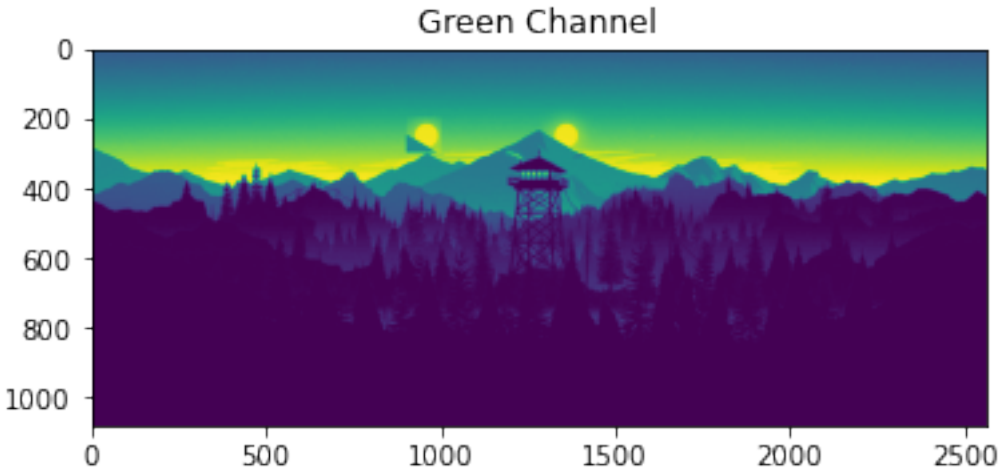
[38]:
```
plt.imshow(b)
plt.title("Blue Channel")
```

[38]: Text(0.5, 1.0, 'Blue Channel')



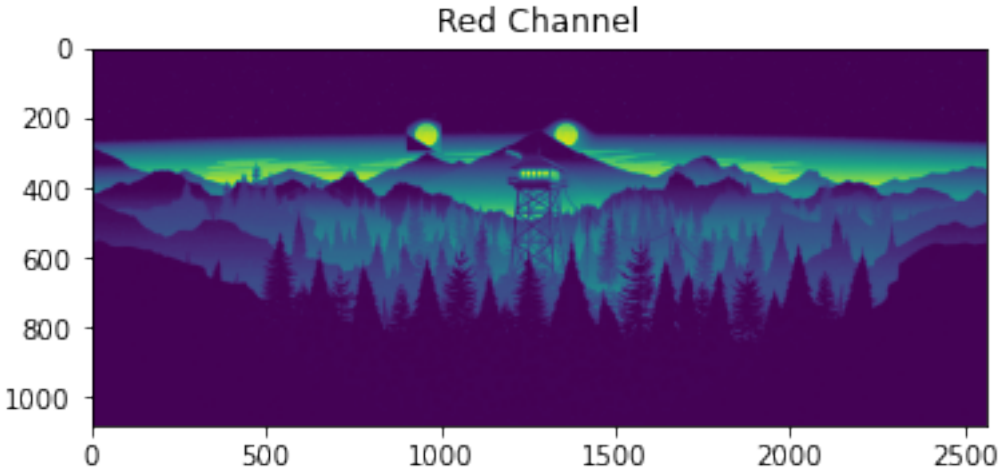
[39]:
```
plt.imshow(g)
plt.title("Green Channel")
```

[39]: Text(0.5, 1.0, 'Green Channel')



[40]:
```
plt.imshow(r)
plt.title("Red Channel")
```

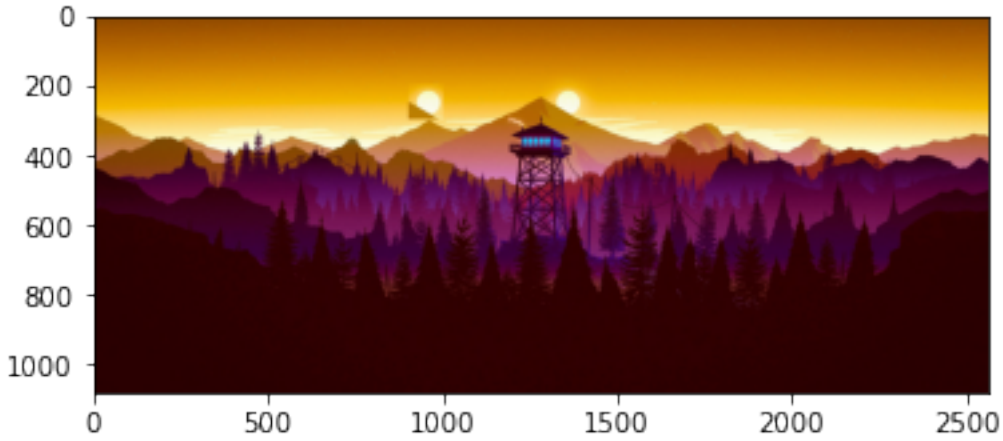
[40]: Text(0.5, 1.0, 'Red Channel')



3.3.2 merge()

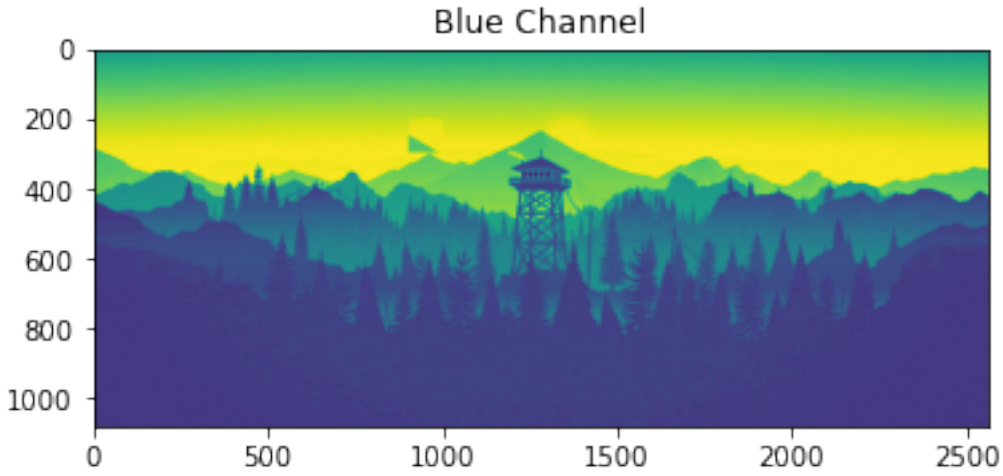
[41]:
```
# merge() permite unir los canales de una imagen
img_unida = cv.merge((b,g,r))
plt.imshow(img_unida)
```

[41]: <matplotlib.image.AxesImage at 0x254b4533e50>



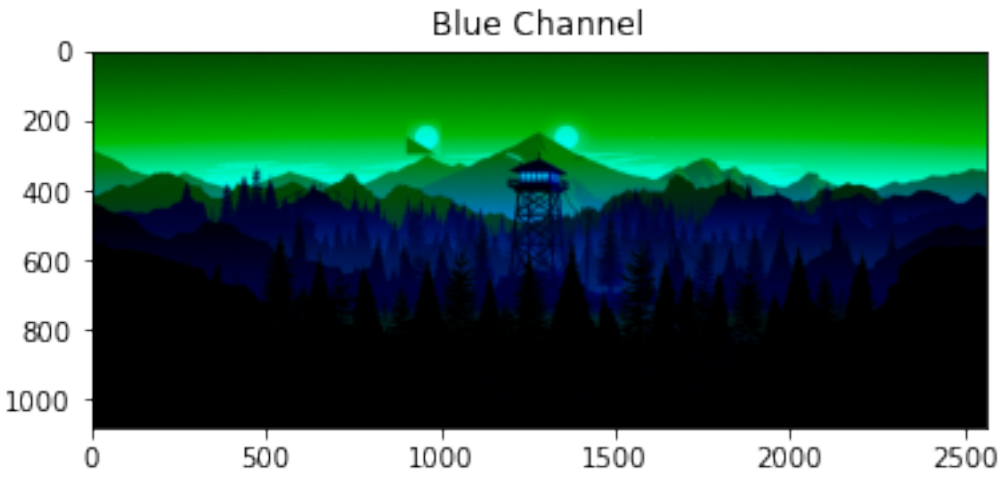
[42]:
```
channel_blue = img[:,:,0]
plt.imshow(channel_blue), plt.title("Blue Channel")
```

[42]: (<matplotlib.image.AxesImage at 0x254b45906a0>, Text(0.5, 1.0, 'Blue Channel'))



[43]:
```
img[:,:,0]=0
plt.imshow(img), plt.title("Blue Channel")
```

[43]: (<matplotlib.image.AxesImage at 0x254b45dfd90>, Text(0.5, 1.0, 'Blue Channel'))



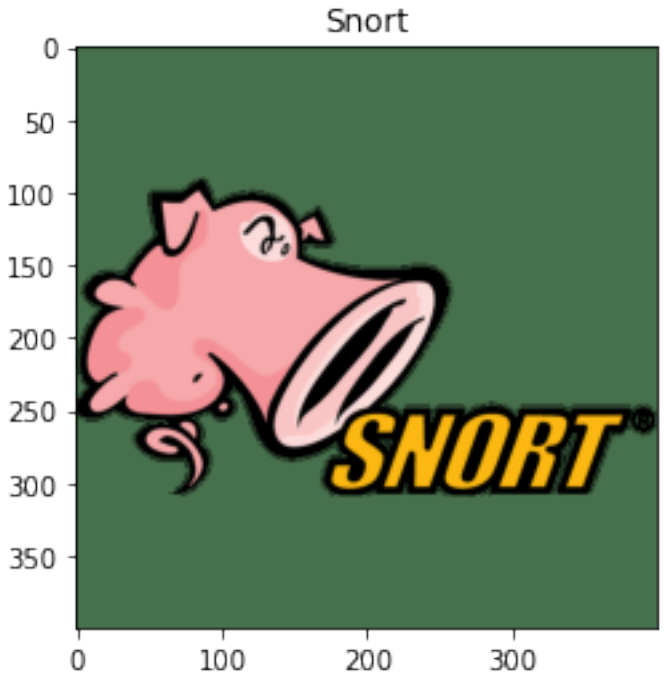
0.6.4 3.4 Operaciones Aritmeticas con Imagenes

3.4.1 Mescla de Imagenes Esto también es una adición de imagen, pero se otorgan diferentes pesos a las imágenes para dar una sensación de fusión o transparencia. Aquí tomé dos imágenes para mezclarlas. La primera imagen tiene un peso de 0.7 y la segunda imagen tiene 0.3.

[44]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread("ImgT2/snort.png")
img2 = cv.imread("ImgT2/wpp1.png")

# cambio de espacio de color
img1 = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

# igualar el tamano
fil,cols,chan = img1.shape
img2 = cv.resize(img2,(cols,fil))
```

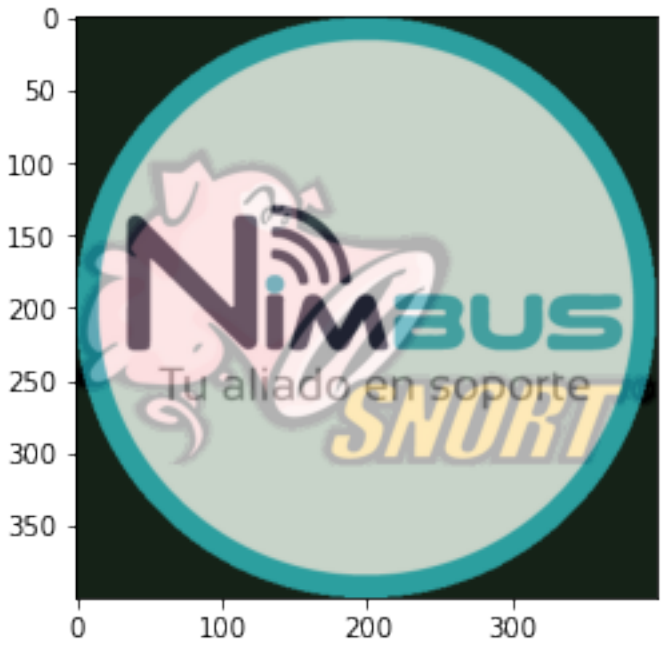
[45]:
```
#Mostramos las imagenes
plt.figure(1)
plt.imshow(img1)
plt.title("Snort")
plt.figure(2)
plt.imshow(img2)
plt.title("Nimbus")
```

[45]: Text(0.5, 1.0, 'Nimbus')



[46]:
```
#combinar imagenes
img_out = cv.addWeighted(img1,0.3,img2,0.7,0)
plt.imshow(img_out)
```

[46]:
```
<matplotlib.image.AxesImage at 0x254b3ea57f0>
```



0.6.5 3.5 Operaciones bit a bit

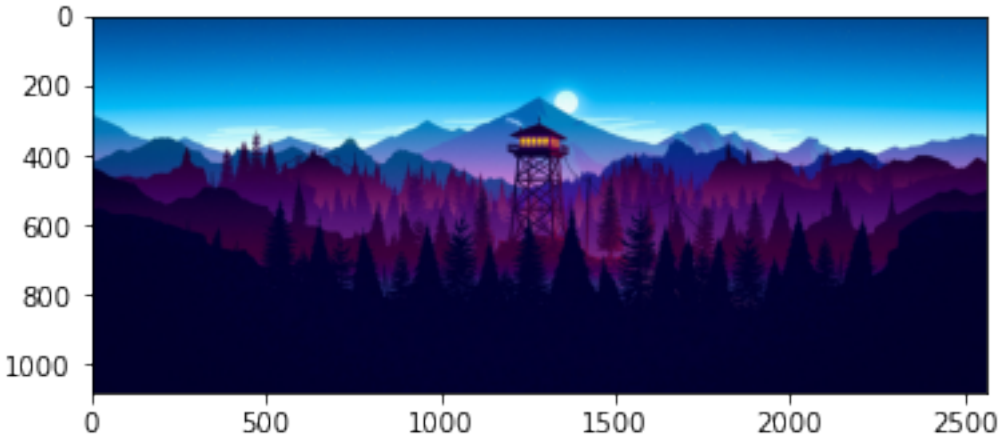
Esto incluye las operaciones bit a bit AND, OR, NOT y XOR. Serán muy útiles al extraer cualquier parte de la imagen (como iremos viendo al transcurso de este aprendisaje), definir y trabajar con ROI no rectangulares, etc. A continuación veremos un ejemplo de cómo cambiar una región particular de una imagen.

Quiero poner el logotipo de OpenCV sobre una imagen. Si agrego dos imágenes, cambiará el color. Si los mezclo, obtengo un efecto transparente. Pero quiero que sea opaco. Si fuera una región rectangular, podría usar el ROI como lo hicimos en el último capítulo. Pero el logotipo de OpenCV no es una forma rectangular. Entonces puede hacerlo con operaciones bit a bit como se muestra a continuación:

[47]:
```
#immportamos las librerias
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# cargar nuestras imagenes
img1 = cv.imread('ImgT2/Horizon Forest.jpg')
```

```
img2 = cv.imread('ImgT2/02.jpg')

# cambiando espacio de color
forest = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
nimbus = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

# mostrado las imagenes leidas
plt.figure(1)
plt.imshow(forest)
plt.figure(2)
plt.imshow(nimbus)
```

[47]: <matplotlib.image.AxesImage at 0x254b7e43310>

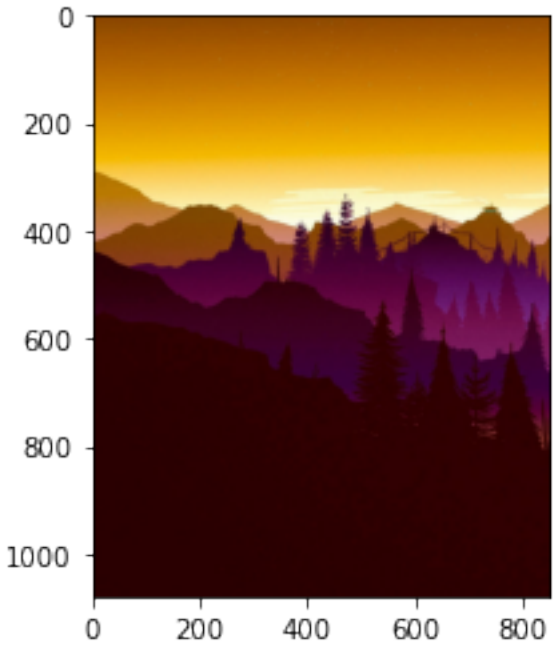


[48]:
```
# cambiamos el tamano de imagen
fil,col,_ = forest.shape
fil2,col2,_ = nimbus.shape
nimbus = cv.resize(nimbus,(col//3,fil))
plt.imshow(nimbus)
```

[48]: <matplotlib.image.AxesImage at 0x254b3efbc70>

[49]:
```
fil, col,_ =nimbus.shape
roi = img1[0:fil,0:col]
plt.imshow(roi)
```

[49]: <matplotlib.image.AxesImage at 0x254c896c3d0>



[50]:
```
# creamos una mascara del logotipo
nimbus_gray = cv.cvtColor(nimbus, cv.COLOR_RGB2GRAY)
ret,mask = cv.threshold(nimbus_gray, 150,255, cv.THRESH_BINARY)
plt.imshow(mask, cmap="gray")
```

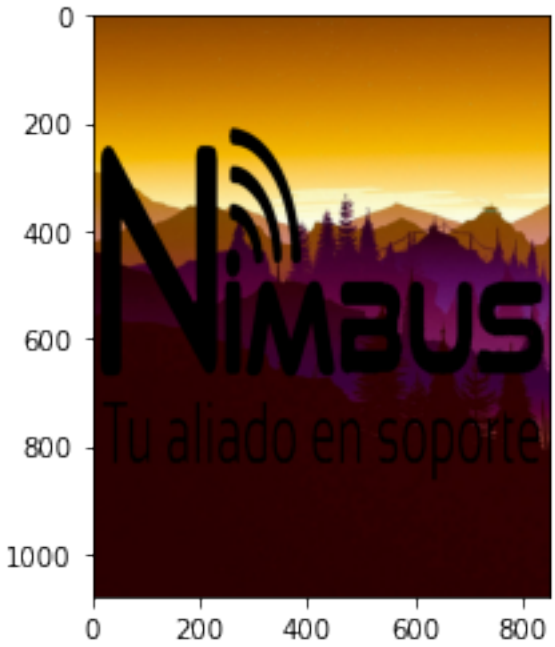
[50]: <matplotlib.image.AxesImage at 0x254c86b71c0>

[51]:
```
# Creamos una mascara invertida
mask_inv = cv.bitwise_not(mask)
plt.imshow(mask_inv, cmap="gray")
```

[51]: <matplotlib.image.AxesImage at 0x254c8708df0>

[52]:
```
# tomamos el roi menos la mascara
img1_bg = cv.bitwise_and(roi,roi,mask = mask)
plt.imshow(img1_bg)
```

[52]: <matplotlib.image.AxesImage at 0x254c8763d60>

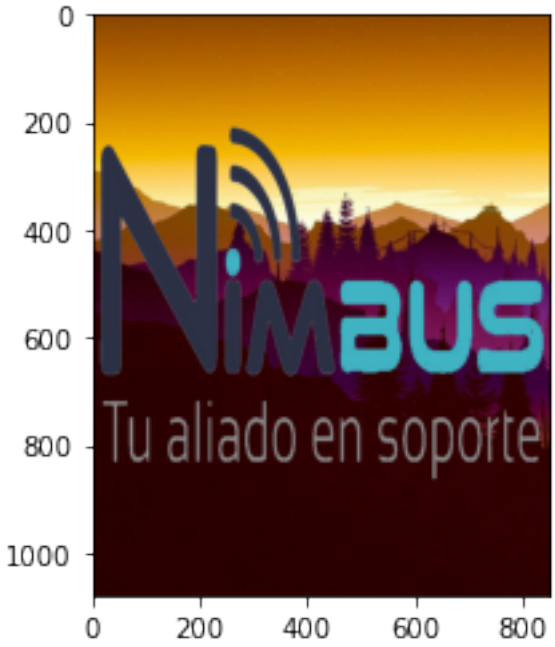


[53]:
```
#tomamos region de intere del logotipo opencv
img2_bg = cv.bitwise_and(nimbus,nimbus,mask=mask_inv)
plt.imshow(img2_bg)
```

[53]: <matplotlib.image.AxesImage at 0x254c87bbc70>

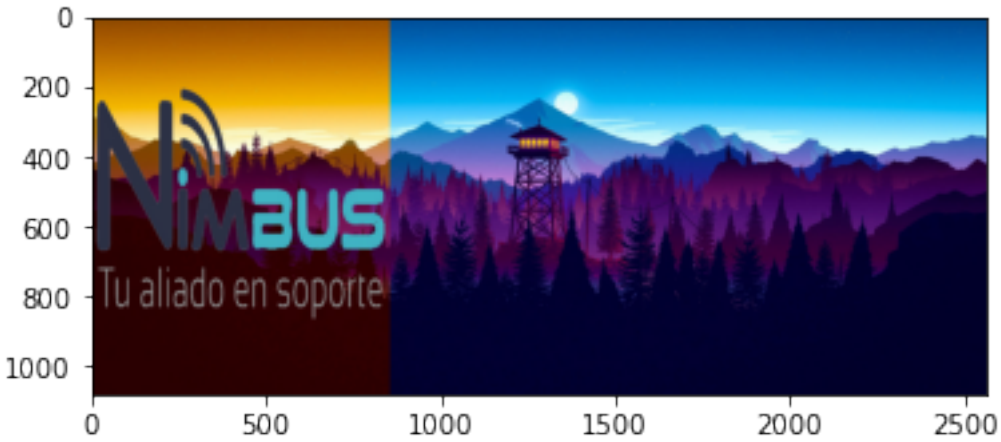
[54]:
```
img = img1_bg + img2_bg
plt.imshow(img)
```

[54]: <matplotlib.image.AxesImage at 0x254c8816a60>



[55]:
```
forest[0:fil,0:col]=img
plt.imshow(forest)
```

[55]: <matplotlib.image.AxesImage at 0x254c8871b20>



0.7 4 Procesamiento de Imagenes

0.7.1 4.1 Cambio de Espacio de Color de las Imagenes

En la visión por computadora y el procesamiento de imágenes, el espacio de color se refiere a una forma específica de organizar los colores. Un espacio de color es en realidad una combinación de dos cosas, un modelo de color y una función de mapeo. La razón por la que queremos modelos de color es porque nos ayuda a representar valores de píxeles usando tuplas. La función de mapeo asigna el modelo de color al conjunto de todos los colores posibles que se pueden representar. Hay muchos espacios de color diferentes que son útiles. Algunos de los espacios de color más populares son RGB, YUV, HSV, Lab, etc. Diferentes espacios de color ofrecen diferentes ventajas. Solo tenemos que elegir el espacio de color adecuado para el problema dado. Tomemos un par de espacios de color y veamos qué información proporcionan:

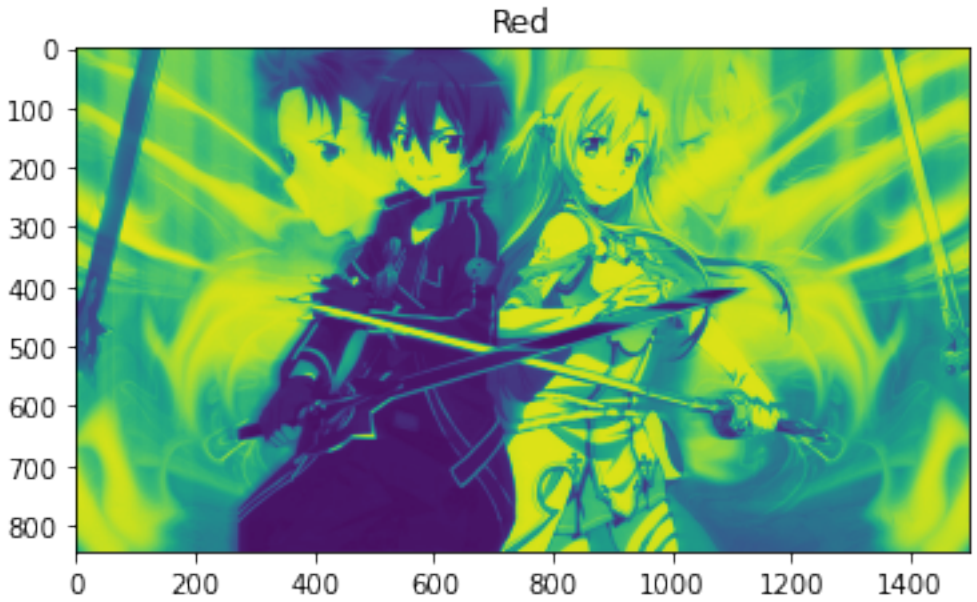
RGB: Probablemente el espacio de color más popular. Es sinónimo de rojo, verde y azul. En este espacio de color, cada color se representa como una combinación ponderada de rojo, verde y azul. Por lo tanto, cada valor de píxel se representa como una tupla de tres números correspondientes a rojo, verde y azul. Cada valor oscila entre 0 y 255.

[56]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("ImgT2/01.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img),plt.title("Imagen Original")
```

[56]:
```
(<matplotlib.image.AxesImage at 0x254c88cd5b0>,
 Text(0.5, 1.0, 'Imagen Original'))
```

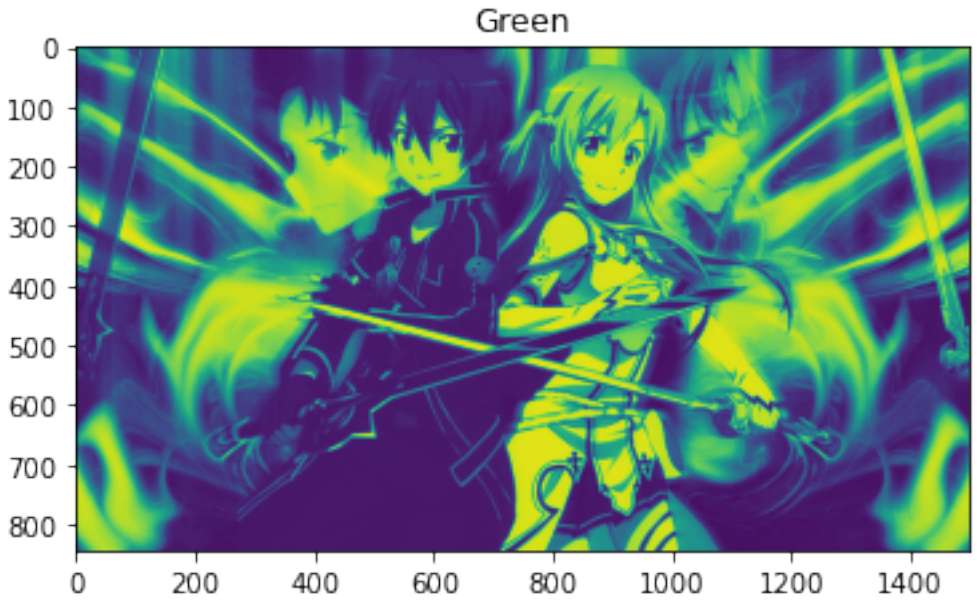
[57]:
```
plt.imshow(img[:,:,0]),plt.title("Red")
```

[57]: (<matplotlib.image.AxesImage at 0x254ca6ff4c0>, Text(0.5, 1.0, 'Red'))



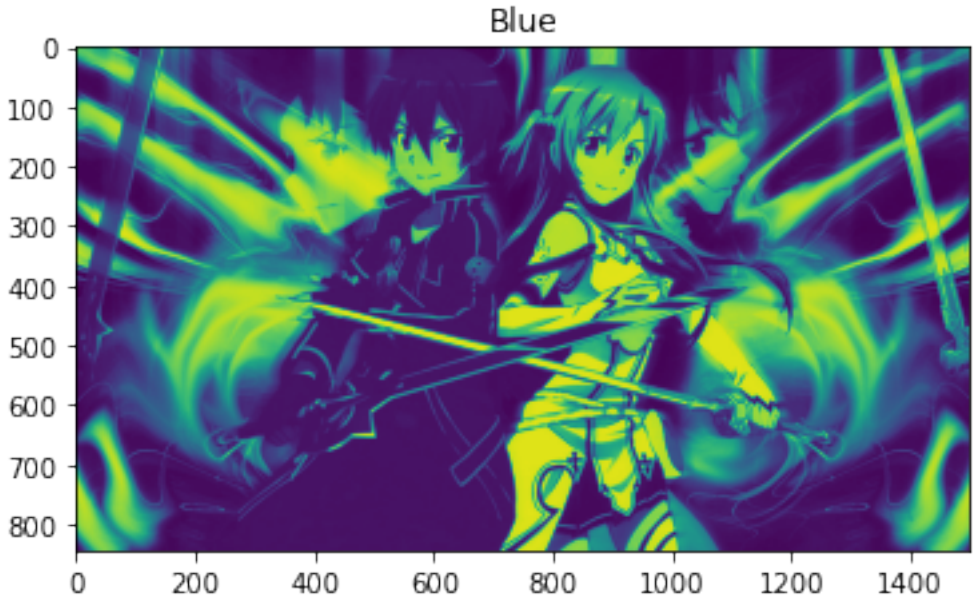
[58]:
```
plt.imshow(img[:,:,1]),plt.title("Green")
```

[58]: (<matplotlib.image.AxesImage at 0x254ca8ae700>, Text(0.5, 1.0, 'Green'))



[59]:
```
plt.imshow(img[:,:,2]),plt.title("Blue")
```

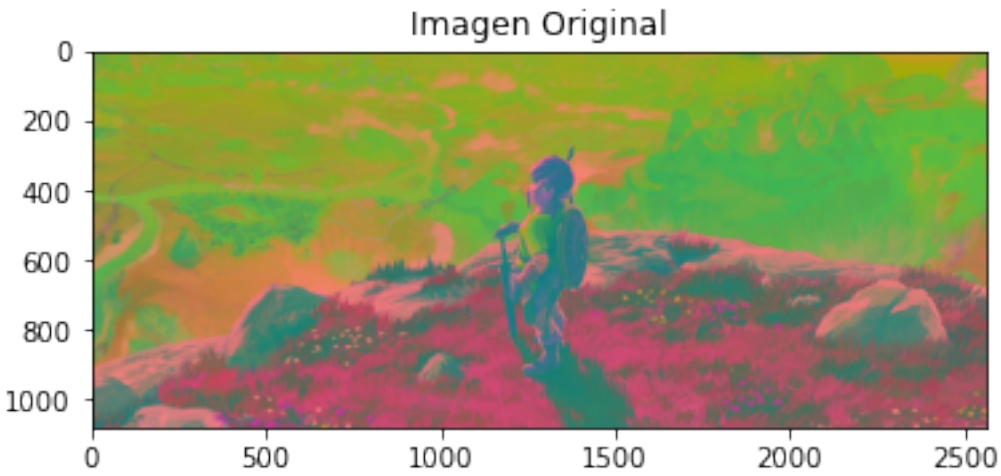
[59]:
```
(<matplotlib.image.AxesImage at 0x254b7e34070>, Text(0.5, 1.0, 'Blue'))
```



YUV: Aunque RGB es bueno para muchos propósitos, tiende a ser muy limitado para muchas aplicaciones de la vida real. La gente comenzó a pensar en diferentes métodos para separar la información de intensidad, de la información de color. Por lo tanto, se les ocurrió el espacio de color YUV. Y se refiere a la luminancia o intensidad, y los canales U / V representan información de color. Esto funciona bien en muchas aplicaciones porque el sistema visual humano percibe la información de intensidad de manera muy diferente a la información de color.

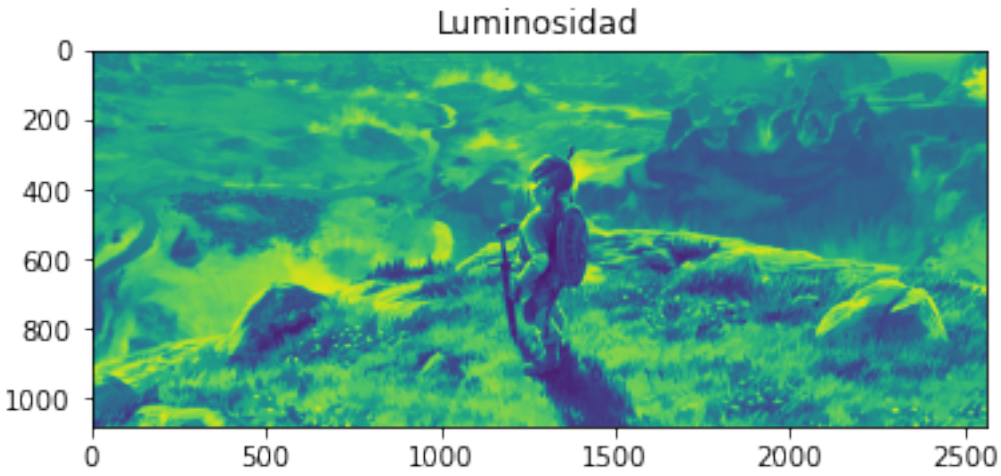
[60]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("ImgT2/Legends of Zelda.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2YUV)
plt.imshow(img),plt.title("Imagen Original")
```

[60]:
```
(<matplotlib.image.AxesImage at 0x254ca715b80>,
 Text(0.5, 1.0, 'Imagen Original'))
```



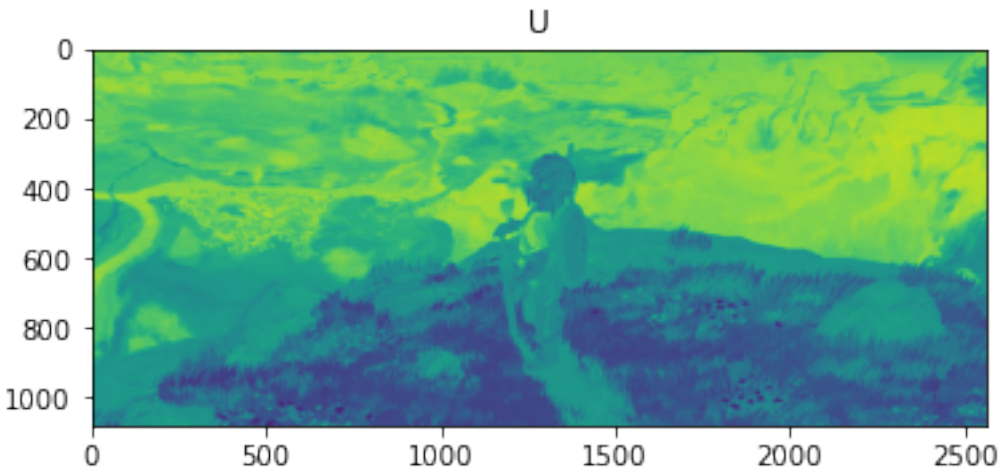
[61]:
```
plt.imshow(img[:,:,0]),plt.title("Luminosidad")
```

[61]: (<matplotlib.image.AxesImage at 0x254cb47d550>, Text(0.5, 1.0, 'Luminosidad'))



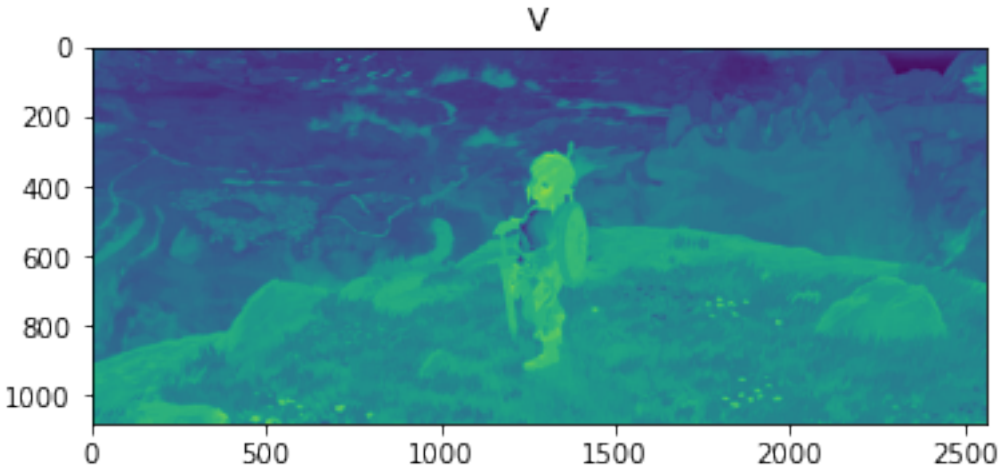
[62]:
```
plt.imshow(img[:,:,1]),plt.title("U")
```

[62]: (<matplotlib.image.AxesImage at 0x254cb78ec70>, Text(0.5, 1.0, 'U'))



[63]:
```
plt.imshow(img[:,:,2]),plt.title("V")
```

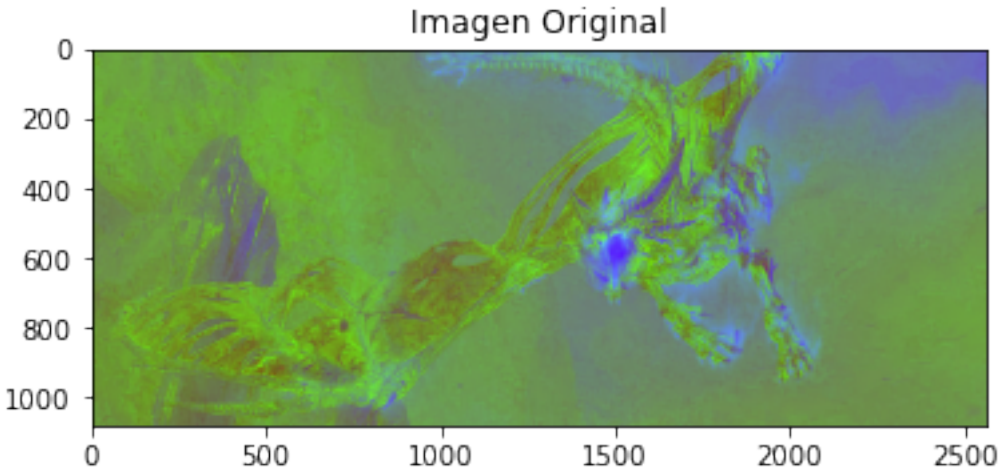
[63]:
```
(<matplotlib.image.AxesImage at 0x254cc90a3a0>, Text(0.5, 1.0, 'V'))
```



HSV: Al final resultó que, incluso YUV todavía no era lo suficientemente bueno para algunos aplicaciones. Entonces la gente comenzó a pensar en cómo los humanos perciben el color, y se les ocurrió el espacio de color HSV. HSV significa Hue, Saturación y Valor(Matiz, Saturacion, Valor). Este es un sistema cilíndrico donde separamos tres de las propiedades más primarias de los colores y las representamos usando diferentes canales. Esto está estrechamente relacionado con la forma en que el sistema visual humano comprende el color. Esto nos da mucha flexibilidad en cuanto a cómo podemos manejar las imágenes.

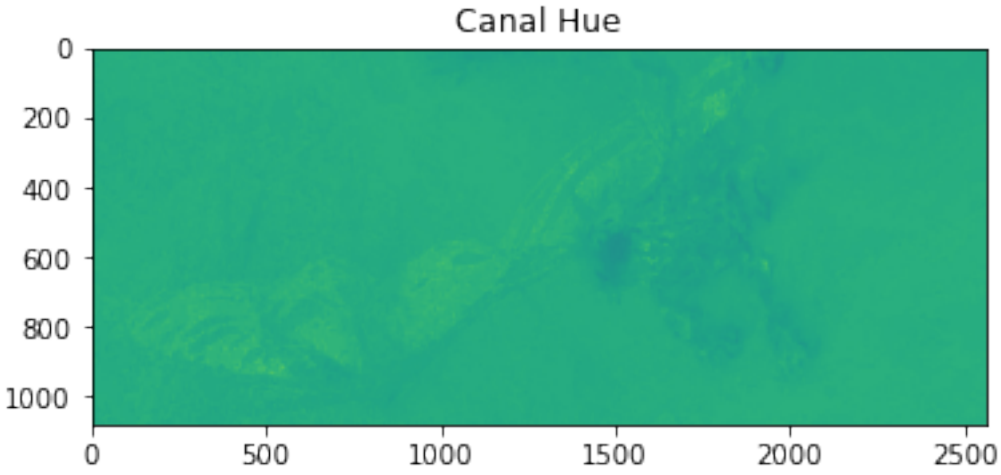
[64]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("ImgT2/Magic Flyer.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
plt.imshow(img),plt.title("Imagen Original")
```

[64]:
```
(<matplotlib.image.AxesImage at 0x254cd40aee0>,
 Text(0.5, 1.0, 'Imagen Original'))
```



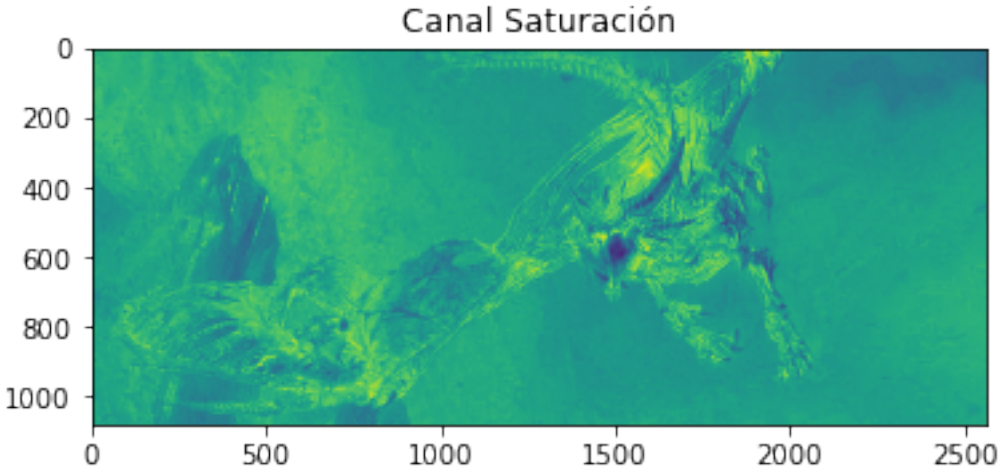
[65]:
```
plt.imshow(img[:,:,0]),plt.title("Canal Hue")
```

[65]:
```
(<matplotlib.image.AxesImage at 0x254cdc684c0>, Text(0.5, 1.0, 'Canal Hue'))
```



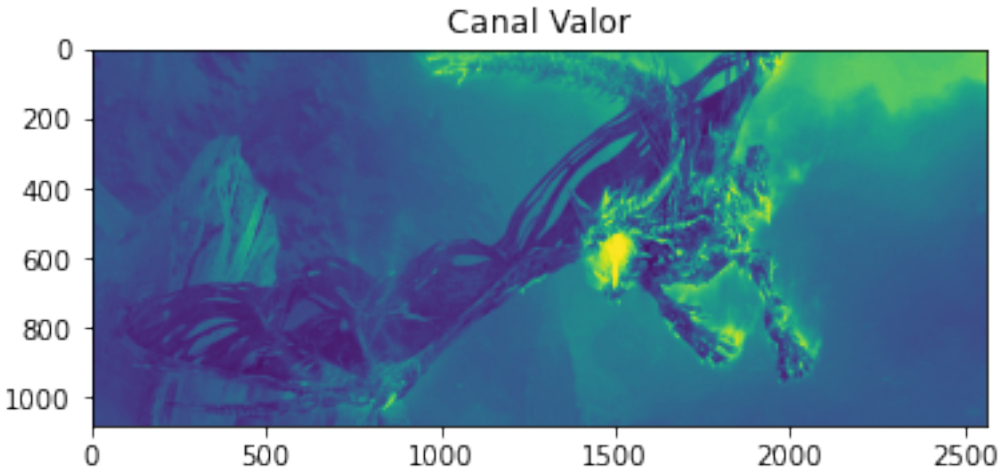
[66]:
```
plt.imshow(img[:,:,1]),plt.title("Canal Saturación")
```

[66]:
```
(<matplotlib.image.AxesImage at 0x254cdf6a970>,
 Text(0.5, 1.0, 'Canal Saturación'))
```



[67]:
```
plt.imshow(img[:,:,2]),plt.title("Canal Valor")
```

[67]:
```
(<matplotlib.image.AxesImage at 0x254ce278160>, Text(0.5, 1.0, 'Canal Valor'))
```



0.8 Tarea 2

Realizar 3 ejercicios:

1. Fusión de imágenes con cv.addWeighted()

2. Extracción del roi (porción de una imagen) y ubicarla en otro lugar de la misma
3. Ejercicio con operaciones Bit a Bit como el visto en clase

0.8.1 Ejercicio 1

Fusión de imágenes con cv.addWeighted()

[68]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread("ImgT2/01.png")
img2 = cv.imread("ImgT2/02.png")

# cambio de espacio de color
img1 = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

# igualar el tamano
fil,cols,chan = img1.shape
img2 = cv.resize(img2,(cols,fil))

#Mostramos las imagenes
plt.figure(1)
plt.imshow(img1)
plt.title("Imagen 1")
plt.figure(2)
plt.imshow(img2)
plt.title("Imagen 2")

#combinar imagenes
img_out = cv.addWeighted(img1,0.3,img2,0.7,0)
plt.figure(3)
plt.imshow(img_out)
plt.title("Imagenes Fusionadas")
```

[68]:
```
Text(0.5, 1.0, 'Imagenes Fusionadas')
```

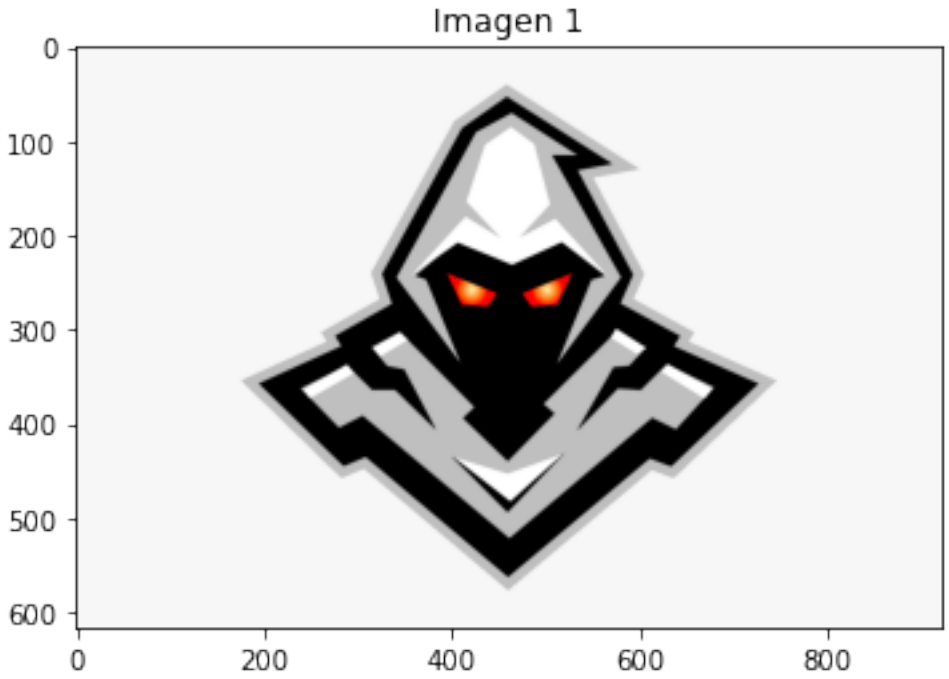


Imagen 1
0
100
200
300
400
500
600
0
200
400
600
800

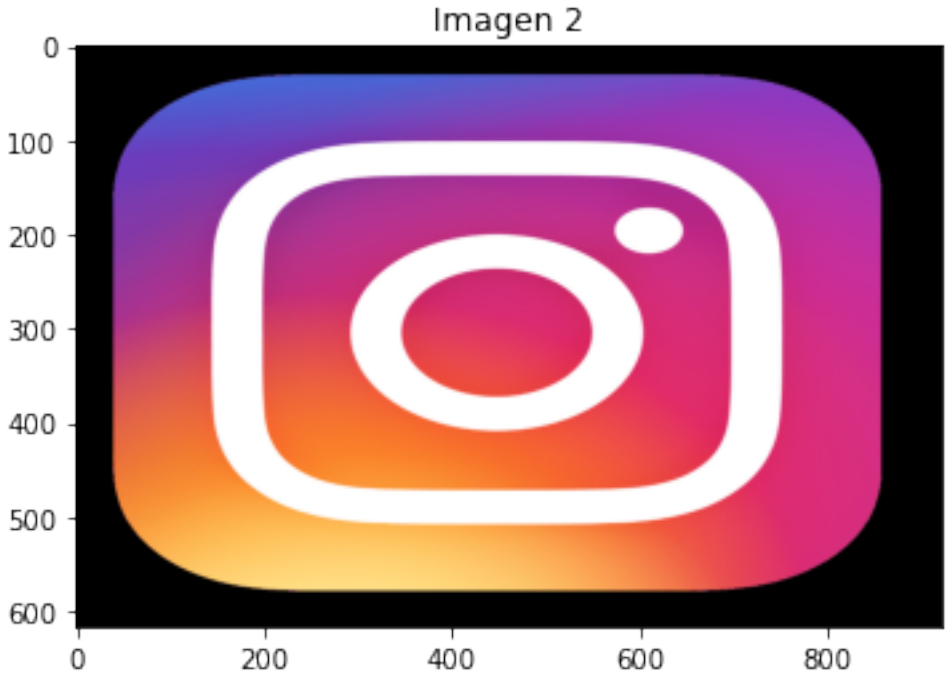
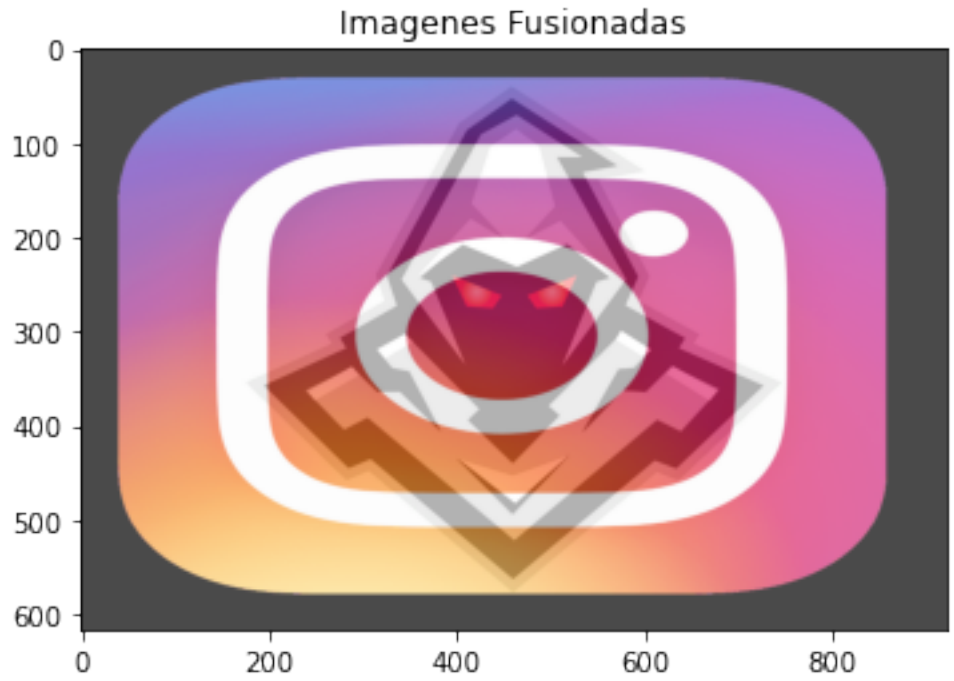


Imagen 2
0
100
200
300
400
500
600
0
200
400
600
800



0.8.2 Ejercicio 2

Extracción del roi (porción de una imagen) y ubicarla en otro lugar de la misma

[69]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("ImgT2/Overwatch 3.jpg")
plt.figure(1)
plt.imshow(img)
plt.title("Imagen Original")

# extraer la region de interes
logo = img[500:900,:700]
plt.figure(2)
plt.imshow(logo)
plt.title("Logo de imagen")

# ubicar en otro lugar de la imagen
img[:400, 1600:2300]=logo
plt.figure(3)
plt.imshow(img)
plt.title("Nueva imagen")
```

[69]:
```
Text(0.5, 1.0, 'Nueva imagen')
```

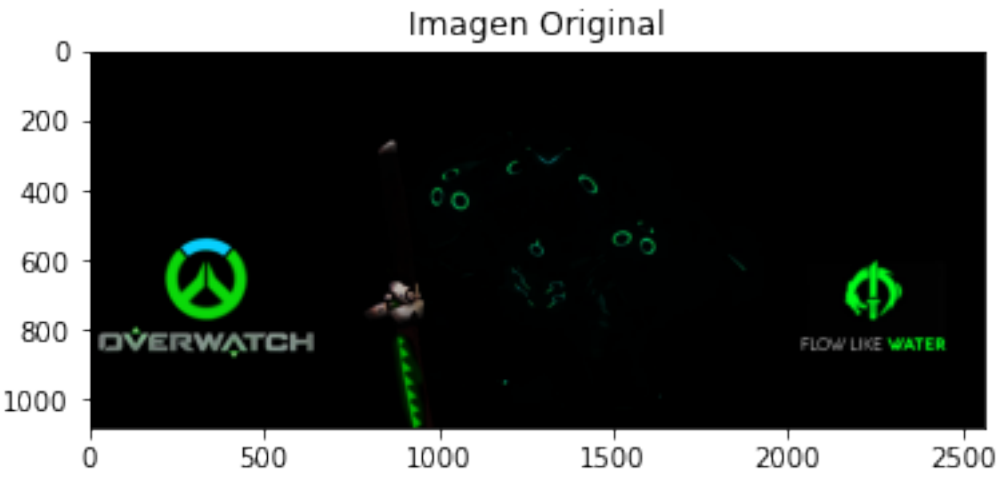
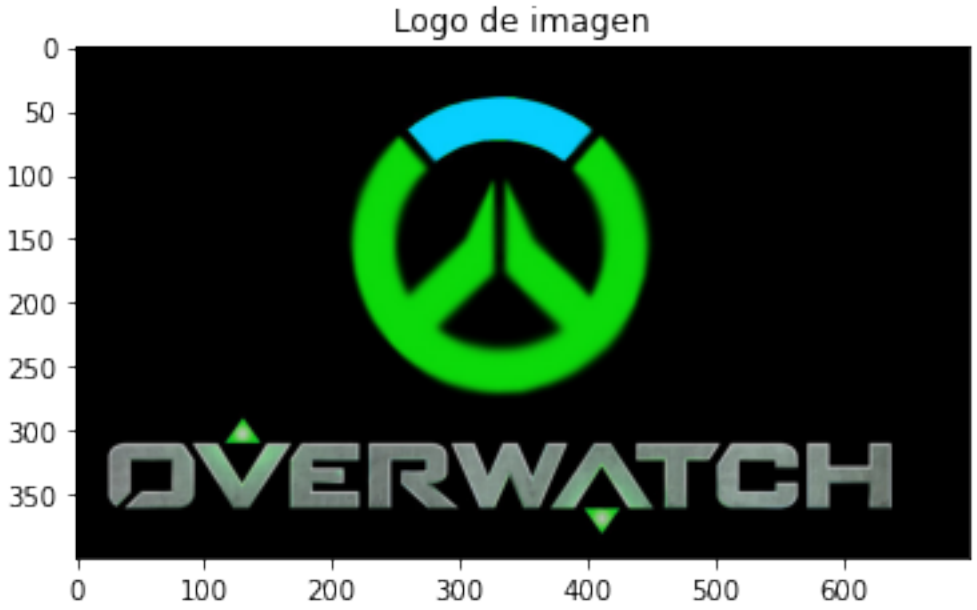
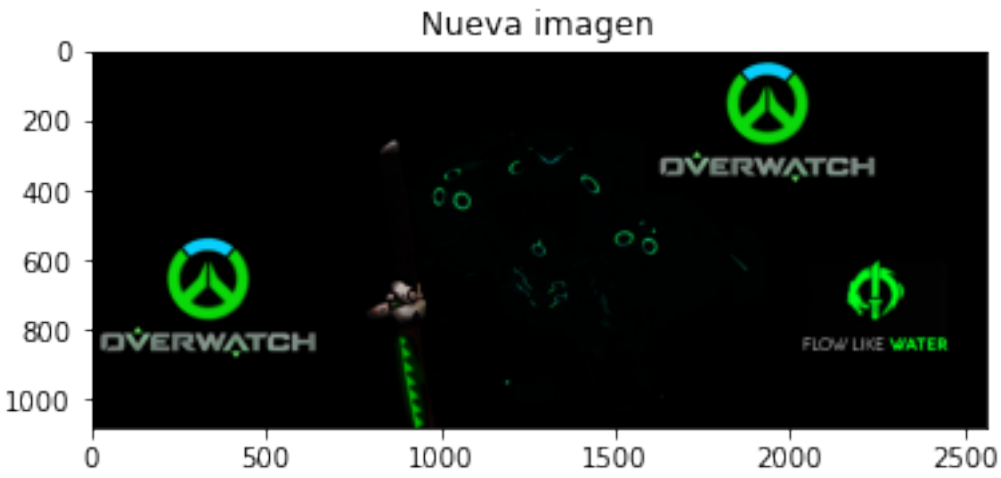


Imagen Original
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
OVERWATCH
FLOW LIKE WATER


Logo de imagen
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600
OVERWATCH

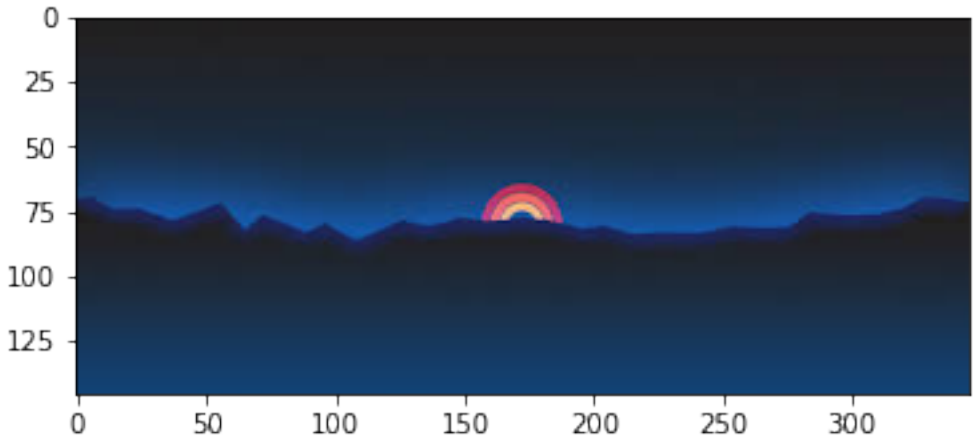


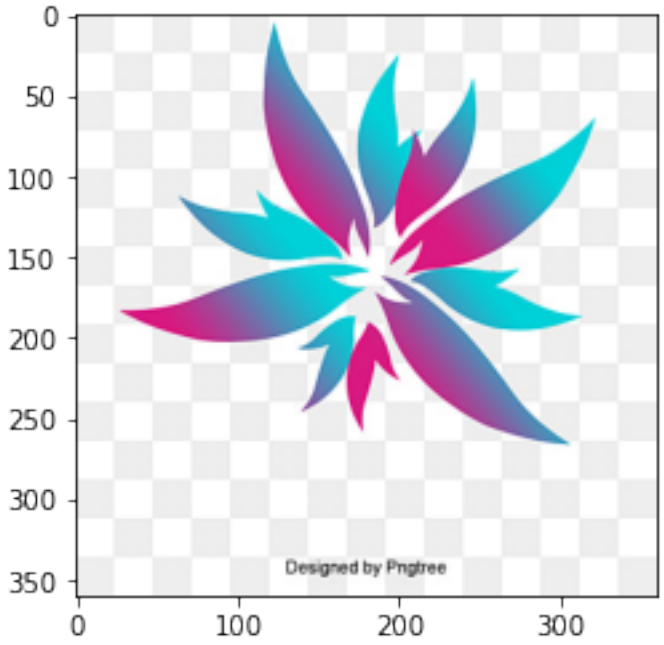
0.8.3 Ejercicio 3

Ejercicio con operaciones Bit a Bit como el visto en clase

[70]:
```
#immportamos las librerias
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# cargar nuestras imagenes
img1 = cv.imread('ImgT2/Rainbow.jpg')
img2 = cv.imread('ImgT2/03.jpg')

# cambiando espacio de color
noche = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
flor = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

# mostrado las imagenes leidas
plt.figure(1)
plt.imshow(noche)
plt.figure(2)
plt.imshow(flor)
```

[70]:
```
<matplotlib.image.AxesImage at 0x254ccd2f460>
```

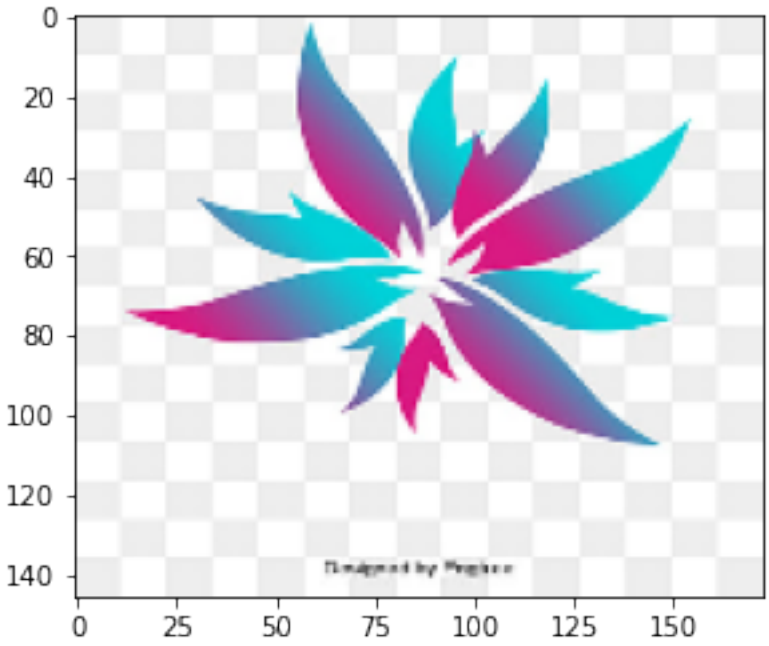


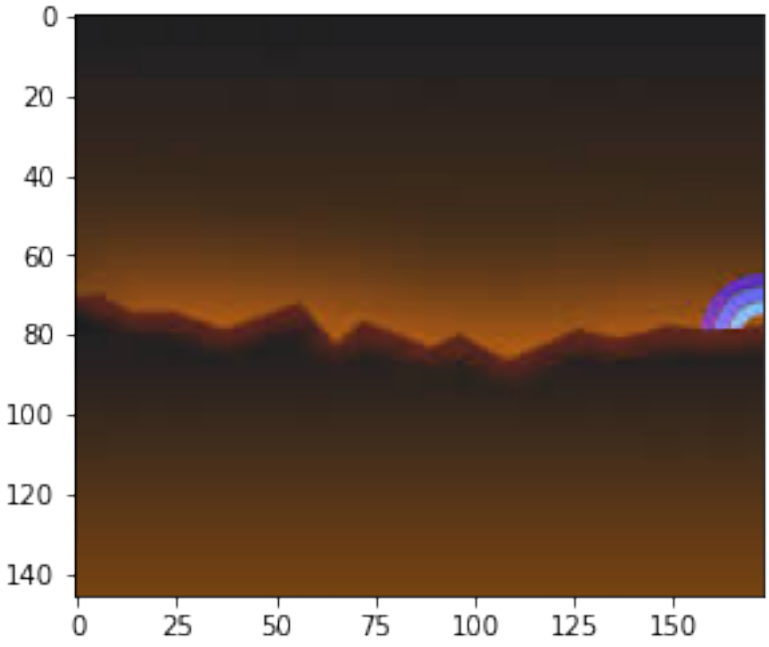


[71]:
```
# cambiamos el tamano de imagen
fil,col,_ = noche.shape
fil2,col2,_ = flor.shape
flor = cv.resize(flor,(col//2, fil))
plt.figure(1)
plt.imshow(flor)

fil, col,_ =flor.shape
roi = img1[0:fil,0:col]
```

```
plt.figure(2)
plt.imshow(roi)
```

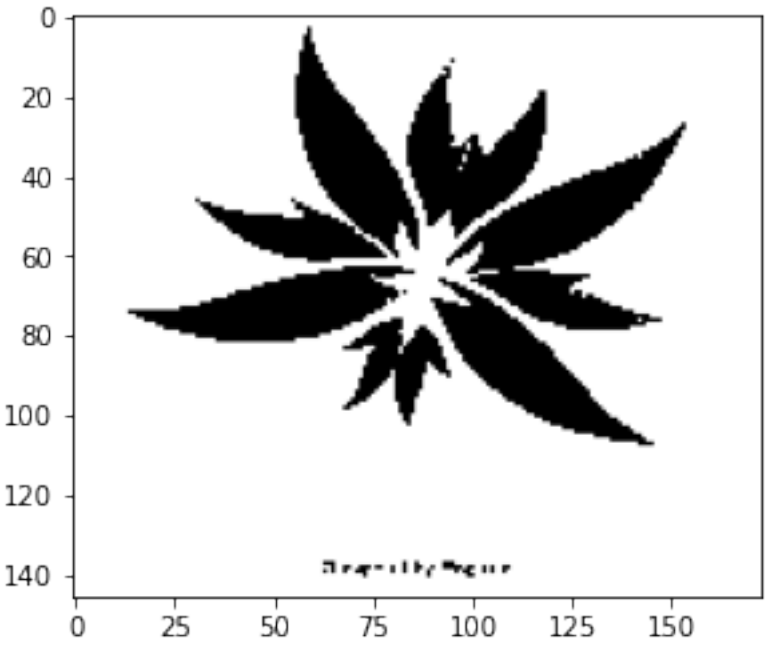
[71]: <matplotlib.image.AxesImage at 0x254ce251730>

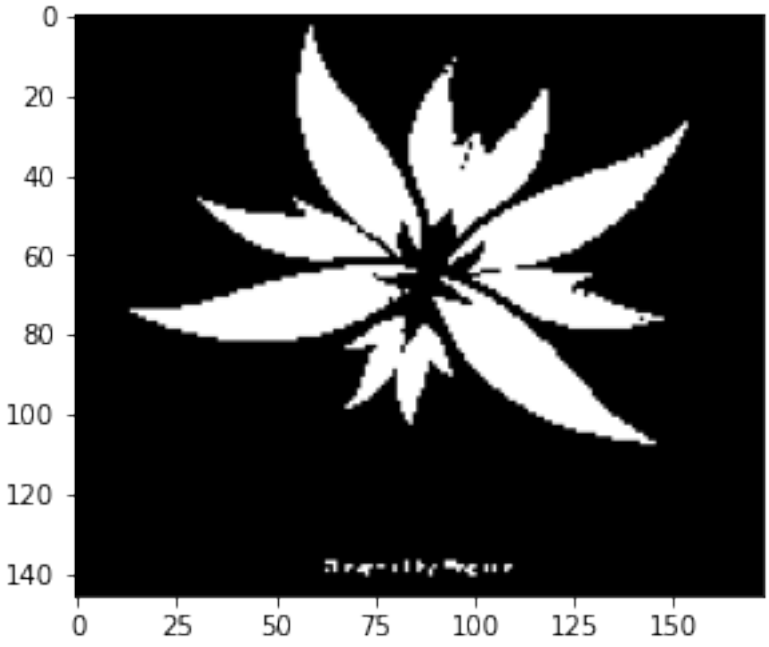


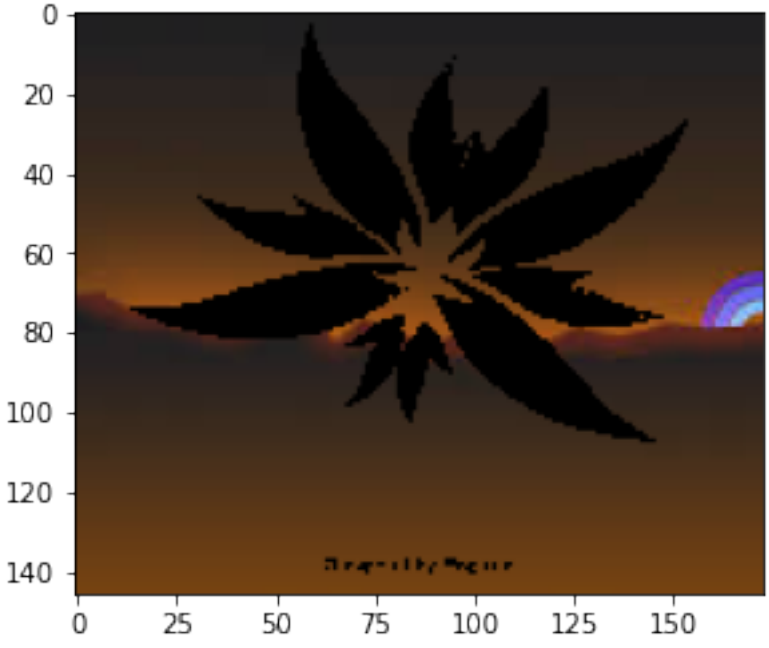


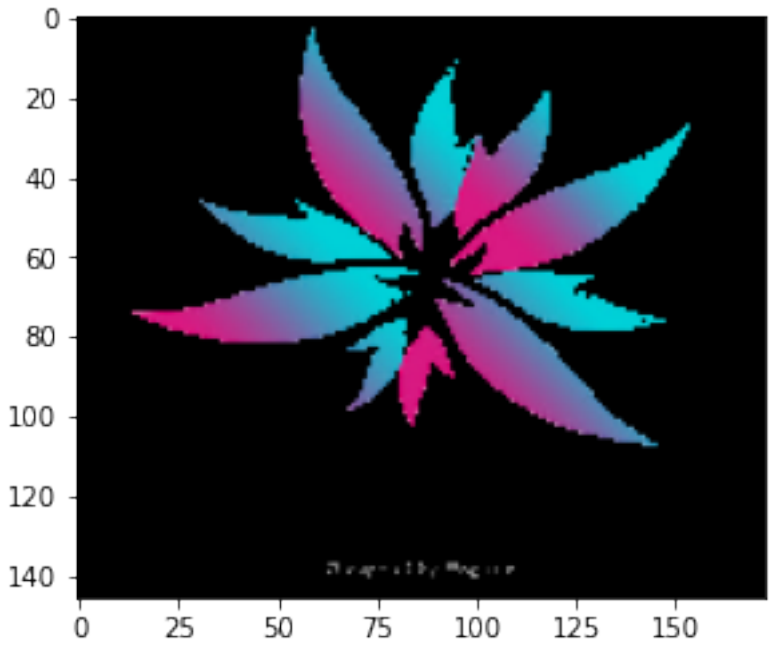
[72]:
```
# creamos una mascara del logotipo
flor_gray = cv.cvtColor(flor, cv.COLOR_RGB2GRAY)
ret,mask = cv.threshold(flor_gray, 150,255, cv.THRESH_BINARY)
plt.figure(1)
plt.imshow(mask, cmap="gray")

# Creamos una mascara invertida
mask_inv = cv.bitwise_not(mask)
plt.figure(2)
plt.imshow(mask_inv, cmap="gray")

# tomamos el roi menos la mascara
img1_bg = cv.bitwise_and(roi,roi,mask = mask)
plt.figure(3)
plt.imshow(img1_bg)

#tomamos region de intere del logotipo
img2_bg = cv.bitwise_and(flor,flor,mask=mask_inv)
plt.figure(4)
plt.imshow(img2_bg)
```

[72]:
```
<matplotlib.image.AxesImage at 0x254cccd7d30>
```



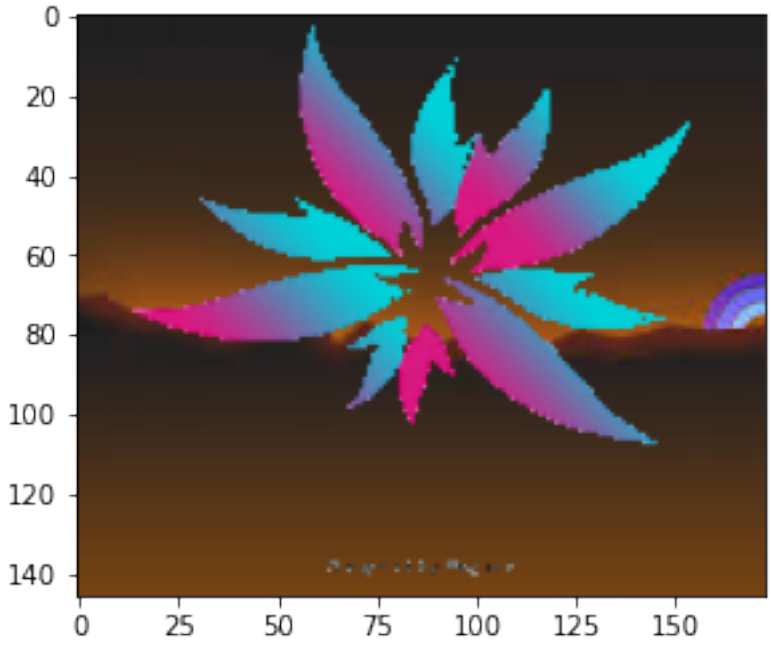

0
20
40
60
80
100
120
140
0
25
50
75
100
125
150

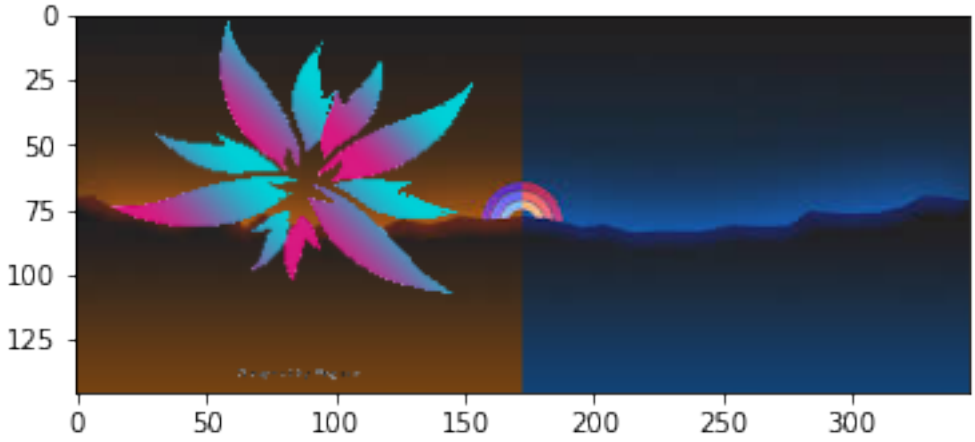

0
20
40
60
80
100
120
140
0
25
50
75
100
125
150



[73]:
```
# Combinar las imagenes
img = img1_bg + img2_bg
plt.figure(1)
plt.imshow(img)

noche[0:fil,0:col]=img
plt.figure(2)
plt.imshow(noche)
```

[73]: <matplotlib.image.AxesImage at 0x254c793cc40>





0.9 Deteccion de Colores

Ahora que sabemos cómo convertir una imagen BGR a HSV, podemos usar esto para extraer un objeto coloreado. En HSV, es más fácil representar un color que en el espacio de color BGR. En nuestra aplicación, intentaremos extraer un objeto de color azul. Así que aquí está el método:

- Toma cada fotograma del video
- Convertir de espacio de color BGR a HSV
- Umbralizamos la imagen HSV para un rango de color azul
- Ahora extraiga el objeto azul solo, podemos hacer lo que queramos en esa imagen

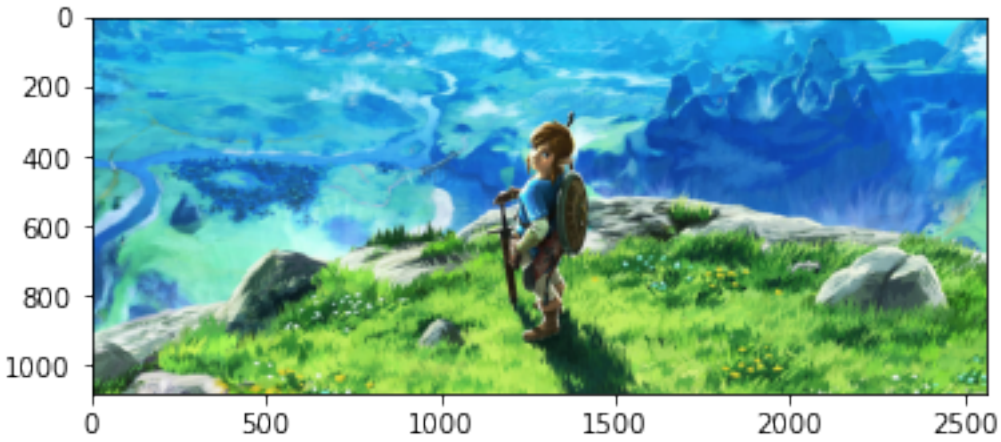
[76]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

cap = cv.VideoCapture(0)

#HSV azul
azul_bajo = np.array([40,100,50], np.uint8)
azul_alto = np.array([80,255,255], np.uint8)

while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        img_hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        mask = cv.inRange(img_hsv, azul_bajo,azul_alto)
        res = cv.bitwise_and(frame,frame, mask=mask)

        cv.imshow("ORIGINAL", frame)
        cv.imshow("Mascara", mask)
        cv.imshow("Resultado", res)
        if cv.waitKey(10) & 0xFF ==ord('q'):
            break

    else:
        break

cap.release()
cv.destroyAllWindows()
```

0.9.1 4.2 Transformaciones Geometricas con Imagenes

4.2.1 Escalado

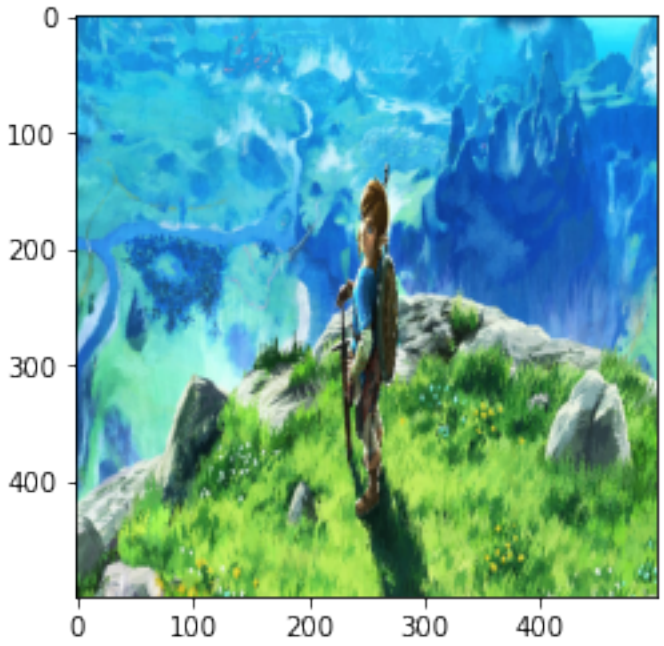
[77]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread("./ImgT3/Legends of Zelda.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img)
```

[77]:
```
<matplotlib.image.AxesImage at 0x254c78f0490>
```



[78]:
```
h,w,_ = img.shape
result = cv.resize(img,(500,500))
plt.imshow(result)
```

[78]: <matplotlib.image.AxesImage at 0x254c7ad08e0>

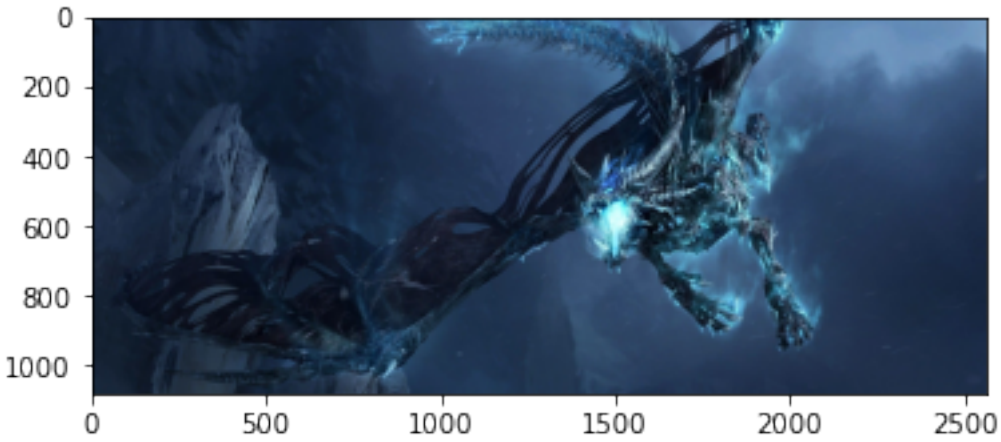


4.2.2 Rotacion

[79]:
```
import numpy as np
import cv2 as cv
```

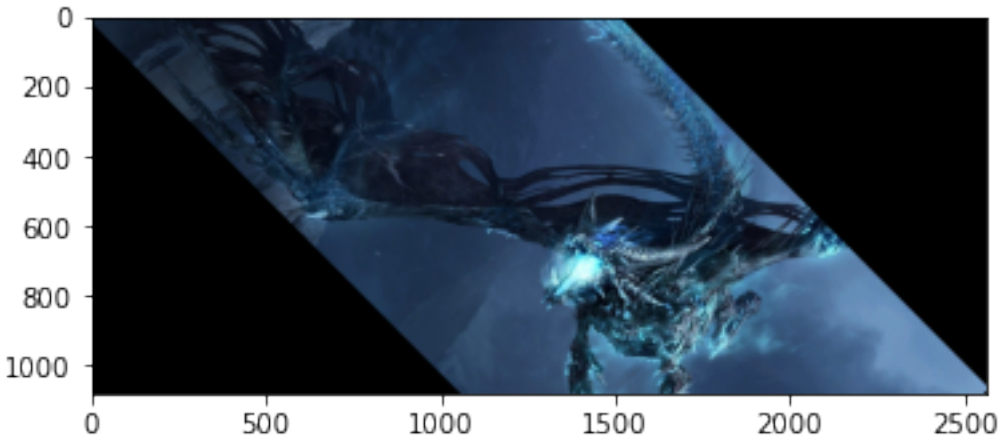
```
import matplotlib.pyplot as plt

img = cv.imread("./ImgT3/Magic Flyer.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img)
```

[79]: <matplotlib.image.AxesImage at 0x254c7b24730>



[80]:
```
# girar la imagen
fil,col,_ = img.shape

# matris de rotacion
m = cv.getRotationMatrix2D(((col-1)/2.0,(fil-1)/2.0),-45,1)

# transformacion de la imagen\
result = cv.warpAffine(img, m, (col,fil))
plt.imshow(result)
```

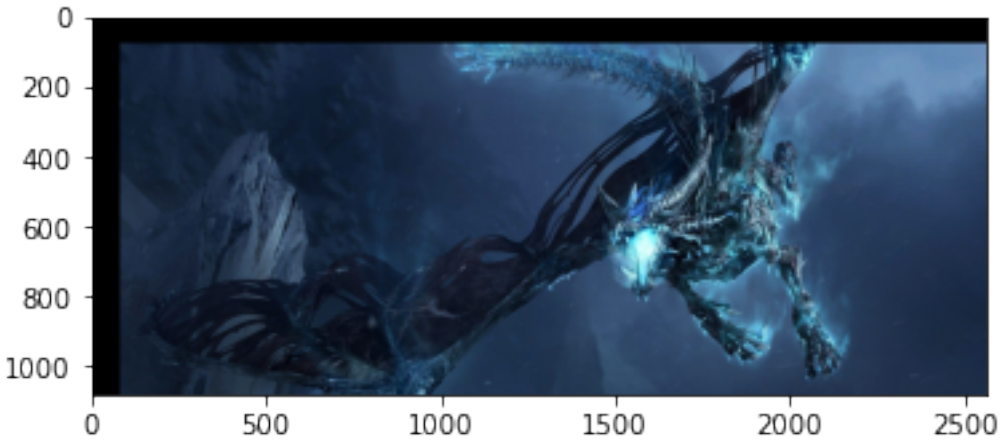
[80]: <matplotlib.image.AxesImage at 0x254c7b74f70>



4.2.3 Traslacion de una imagen

[82]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread("./ImgT3/Magic Flyer.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

fil,col,_ =img.shape
M = np.float32([[1,0,80],[0,1,80]])
result = cv.warpAffine(img, M, (col,fil))
plt.imshow(result)
```

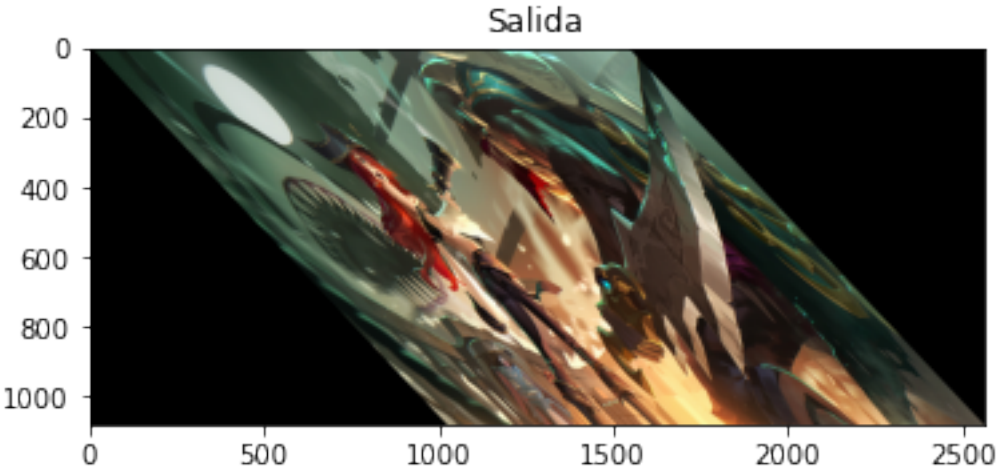
[82]:
```
<matplotlib.image.AxesImage at 0x254c7c2f130>
```



4.2.4 Transformacion Afine

[84]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread("./ImgT3/Miss Fortune.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.figure(1)
plt.imshow(img), plt.title("Original")
fil,col,_ =img.shape

# puntos de entrada
scr_points = np.float32([[0,0], [col-1,0], [0,fil-1]])

#puntos de salida
dts_points = np.float32([[0,0] , [int(0.6*(col-1)),0] , [int(0.
↪4*(col-1)),fil-1]])

# matris de transformacion Afin
matris_Afin = cv.getAffineTransform(scr_points,dts_points)

# Transformacion
output = cv.warpAffine(img,matris_Afin,(col,fil))

# salida
plt.figure(2), plt.imshow(output), plt.title("Salida")
```

[84]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.image.AxesImage at 0x254c7cb8b20>,
 Text(0.5, 1.0, 'Salida'))
```



Espejo de una imagen

[85]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("ImgT3/Overwatch 1.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.figure(1)
plt.imshow(img), plt.title("Original")
fil,col,_ =img.shape
# puntos de entrada
scr_points = np.float32([[0,0], [col-1,0], [0,fil-1]])
#puntos de salida
dts_points = np.float32([[col-1,0], [0,0],[col-1, fil-1]])
# matris de transformacion Afin
matris_Afin = cv.getAffineTransform(scr_points,dts_points)
# Transformacion
output = cv.warpAffine(img,matris_Afin,(col,fil))
# salida
plt.figure(2), plt.imshow(output), plt.title("Salida")
```

[85]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.image.AxesImage at 0x254c7d12820>,
 Text(0.5, 1.0, 'Salida'))
```

Trasfomaciones de Perspectiva

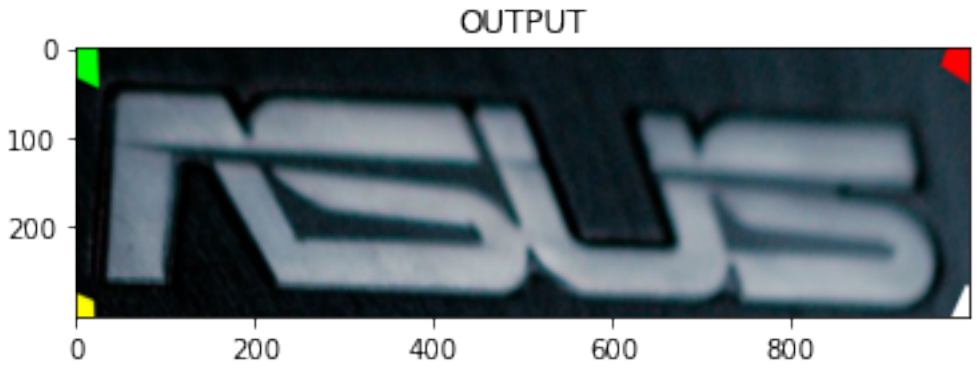
[131]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img=cv.imread("ImgT3/Asus.jpg")
img = cv.cvtColor(img,cv.COLOR_BGR2RGB)

# arriba izquierda verda
img[350:380,600:630]=[0,255,0]

#arriba derecha rojo
img[130:160,1210:1240]=[255,0,0]
```

```
# abajo izquierda amarillo
img[500:530,590:620]=[255,255,0]

# abajo derecha blanco
img[250:280,1300:1330]=[255,255,255]

plt.imshow(img)
```

[131]: <matplotlib.image.AxesImage at 0x254cd0db7c0>

[135]:
```
# Coordenadas X, Y
pts1 = np.float32([[615,365],[1225,145],[605,515],[1315,265]])
pts2 = np.float32([[0,0],[1000,0],[0,300],[1000,300]])

matriz = cv.getPerspectiveTransform(pts1,pts2)
output = cv.warpPerspective(img, matriz,(1000,300))

plt.imshow(output),plt.title("OUTPUT")
```

[135]: (<matplotlib.image.AxesImage at 0x254cd2c90d0>, Text(0.5, 1.0, 'OUTPUT'))



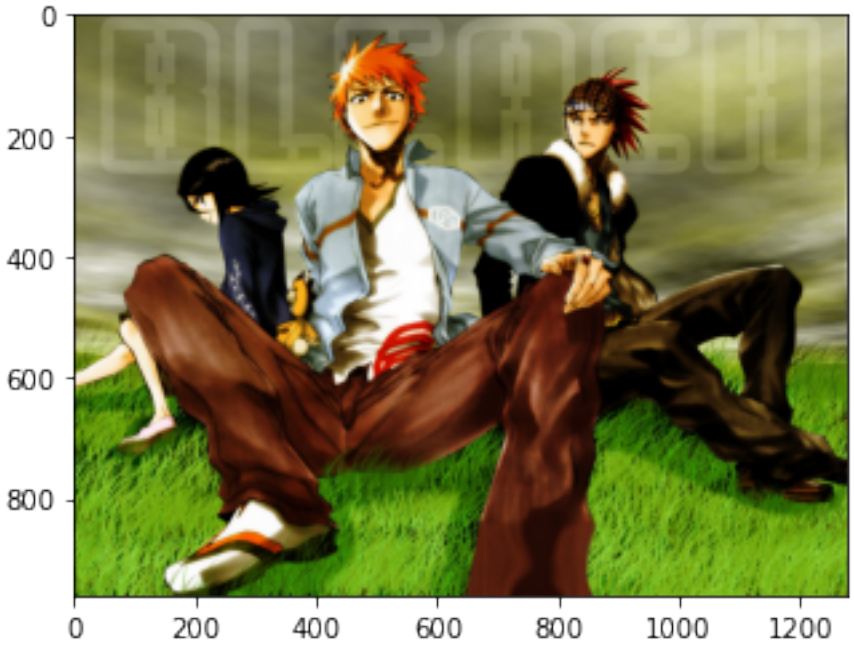
0.9.2 4.3 Umbralizacion de Imagenes

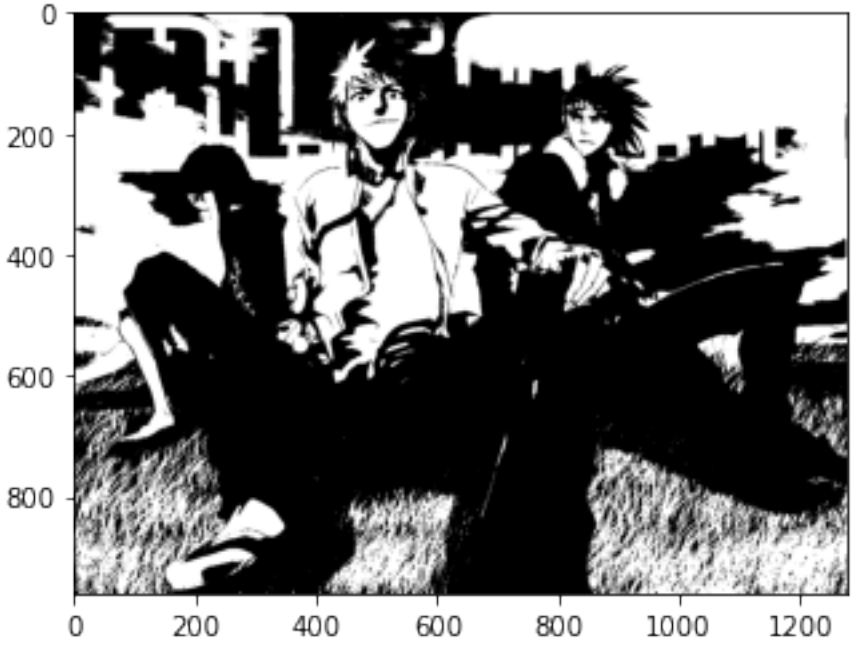
4.3.1 Umbralizacion Simple

[137]:
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('ImgT3/wallpaper-135875.jpg')
img = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

ret,thresh1 = cv.threshold(img,70,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,70,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,70,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,70,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,70,255,cv.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

[138]:
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('ImgT3/wallpaper-175109.jpg')
img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(img,cv.COLOR_RGB2GRAY)
ret,th1 = cv.threshold(img2,127,255,cv.THRESH_BINARY)
plt.figure(1),plt.imshow(img)
plt.figure(2),plt.imshow(th1, cmap="gray")
```

[138]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.image.AxesImage at 0x254ced2b8e0>)
```


0
200
400
600
800
0
200
400
600
800
1000
1200


0
200
400
600
800
0
200
400
600
800
1000
1200

[ ]: