

Taller Nº 1 - Caricaturas mediante Filtros			
Asignatura	Tópicos Especiales - Visión Artificial	Código	0756
Profesor	José Carlos Rangel Ortiz		

## 1. Introducción

Este taller consiste en la aplicación de los conceptos de filtros de OpenCV de manera conjunta con la intención de dar a un imagen la apariencia de ser una caricatura o dibujo realizado con trazos.

En este caso la aplicación de los filtros de manera conjunta realiza cambios acumulativos en la imagen los cual permite crear dicho efecto en la imagen resultante.

## 2. Filtros para Caricaturizar una Imagen

Los pasos para transformar una imagen a un estilo de caricatura se pueden resumir en:

**Paso 1:** Aplicar un **Filtro Bilateral** para reducir la paleta de color de la imagen.

**Paso 2:** Convertir la imagen original a escala de grises.

**Paso 3:** Aplicar un **Median Blur** para reducir el ruido

**Paso 4:** Utilizar **Adaptative Thresholding** (Umbralizado adaptativo), para detectar y enfatizar los bordes en una máscara de bordes.

**Paso 5:** Combinar la imagen de color del **Paso 1** con la máscara de bordes del **Paso 4**.

Seguidamente se presentaran los códigos para cada uno de estos pasos que permitirán aplicar estos filtros a una imagen de entrada.

1. Iniciamos importando las librerías

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

2. Los siguientes fragmentos forman parte de una función en Python, la cual tiene como encabezado la siguiente línea, esta recibe la imagen como un NumPy Array, la cantidad de sub-muestreos a realizar y la cantidad de filtros bilaterales a aplicar, lo cual se explica posteriormente.

```
def cartoonize(rgb_image: np.ndarray, num_pyr_downs=2, num_bilaterals=7):
```

3. Como parte del primer paso se realiza un submuestreo de la imagen original utilizando el enfoque de pirámide de imágenes [Image Pyramid OpenCV](#). Esto nos permite reducir el tamaño de la imagen de entrada.

```
downsampled_img = rgb_image
for _ in range(num_pyr_downs):
    downsampled_img = cv2.pyrDown(downscaled_img)
```

4. Seguidamente aplicamos un conjunto de filtros bilaterales a la imagen, esto se realiza para que el computo sea más efectivo en el ordenador, ya que la aplicación de un filtro bilateral de gran radio, cuesta mucho más que aplicar consecutivamente varios filtros con menor radio. La cantidad de veces que se aplica el filtro puede ser definida por el usuario mediante el parámetro *num\_bilaterals* que se envía a la función.

```
for _ in range(num_bilaterals):
    filterd_small_img = cv2.bilateralFilter(downscaled_img, 9, 9, 7)
```

5. Luego de esto se sobremuestrea (*upsampling*) la imagen filtrada mediante las pirámides de imágenes, para dejar el tamaño original, es este caso al ser mediante pirámides se estará perdiendo información, la cual será generada por OpenCV. Más información sobre las Pirámides de imágenes disponible en [OpenCV Doc](#).

```
filtered_normal_img = filterd_small_img
for _ in range(num_pyr_downs):
    filtered_normal_img = cv2.pyrUp(filtered_normal_img)
```

6. El siguiente fragmento permite verificar que la cantidad de canales de la imagen filtrada sea el mismo que la imagen original.

```
if filtered_normal_img.shape != rgb_image.shape:
    filtered_normal_img = cv2.resize(
        filtered_normal_img, rgb_image.shape[:2])
```

7. Terminado esta primera parte, se procede a iniciar con el Paso 2 de nuestro procedimiento. Para ello se convierte la imagen original a escala de grises, ya que de esta manera se podrán detectar mejor los bordes en próximos pasos.

---

```
img_gray = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY)
```

---

8. Luego de esto se procede al Paso 3, donde se reduce el ruido en nuestra imagen en escala de grises, mediante el uso de un filtro **Median Blur**.

---

```
img.blur = cv2.medianBlur(img_gray, 7)
```

---

9. Como Paso 4 se procede a la aplicación del **Umbralizado Adaptativo** (*Adaptive Threshold*), para enfatizar los bordes y crear una máscara con estos. Esta función utiliza un valor de umbral de pixel, para convertir una imagen en escala de grises a binaria. Por lo cual, si el valor de un pixel esta por encima del umbral, entonces el valor del pixel en la imagen resultante será 255, de lo contrario será 0. Este método es robusto frente a condiciones de iluminación lo cual se busca cuando deseamos marcar las líneas negras gruesas alrededor de los elementos de una imagen.

---

```
gray_edges = cv2.adaptiveThreshold(img.blur, 255,
                                    cv2.ADAPTIVE_THRESH_MEAN_C,
                                    cv2.THRESH_BINARY, 9, 2)
```

---

10. Como Paso 5 se procede a combinar nuestra máscara de bordes del paso anterior, con la imagen resultante luego de aplicar los filtros bilaterales del paso 1. En este caso se aplica una operación lógica **BITWISE AND** de OpenCV, en esta un pixel de una imagen resultante tiene un valor si y solo si ambos pixeles de cada imagen son mayores a 0, esta operación hace que la máscara y la imagen se fundan en una sola, respetando el criterio definido de unión ver Fig. 1. En el caso de imágenes para que un pixel resultado sea blanco (255) ambos pixeles deben tener el valor blanco, de lo contrario aplica el valor más cercano a cero (0). En este caso, tendremos una imagen que contiene solo los bordes detectados y otra con el contenido de la imagen luego de ser pasado a través del filtro bilateral.

TABLA DE VERDAD (AND)		En OpenCV:	Visualización
A	B		
0	0	0	
0	1	0	
1	0	0	
1	1	1	

		A	B	SALIDA
		0	0	0
		0	255	0
		255	0	0
		255	255	255

Figura 1: Tabla de Verdad para Bitwise AND. Tomado de **BITWISE AND**.

---

```
rgb_edges = cv2.cvtColor(gray_edges, cv2.COLOR_GRAY2RGB)
return cv2.bitwise_and(filtered_normal_img, rgb_edges)
```

---

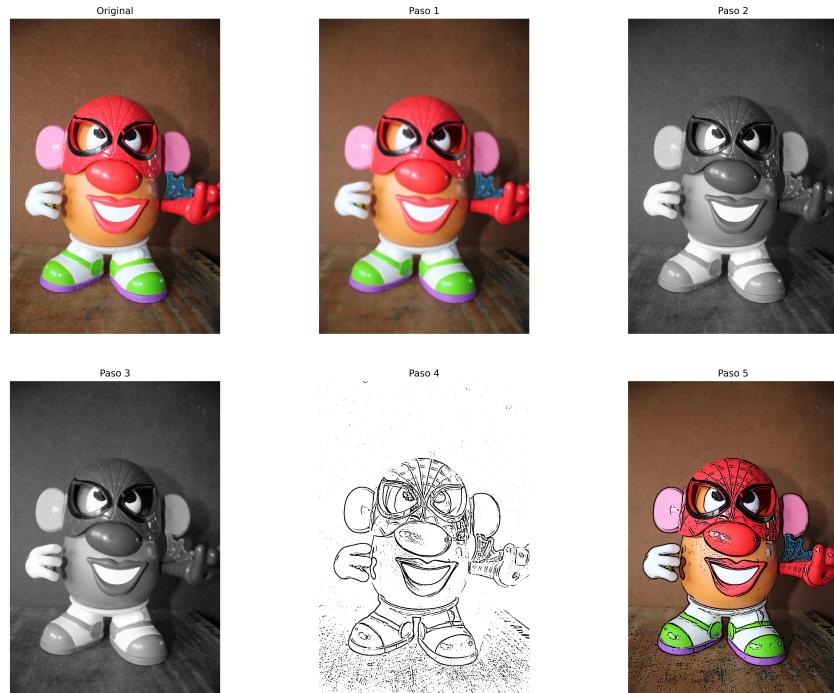


Figura 2: Resultados obtenidos por fase.

### 3. Conversión en Tiempo Real con una Cámara

1. Esta segunda parte incluye el código para utilizar la cámara web, en la instrucción cv2.VideoCapture(0) definimos que cámara utilizaremos en caso de que se tengan varias conectadas al equipo.

---

```
# Capturar desde la cámara web
video_capture = cv2.VideoCapture(0)
frame_index = 0

while True:
    # iniciar captura
    ret, frame = video_capture.read()

    # TODA MODIFICACION A LA IMAGEN DE ENTRADA
    # SE DEBE COLOCAR EN ESTA PARTE DEL CODIGO
    #
    #

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

        cv2.imshow('OpenCV Segmentacion de Color', final_image)
    # PResionar 'q' para salir
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```

---

### **Entregables del taller**

1. Copie los diferentes pasos expresados en la primera parte y codifique la función que los implemente adecuadamente.
2. Muestre cada una de las imágenes resultantes para cada paso del proceso de caricaturizar, utilizando una imagen de su gusto. Puede consultar la imagen [2](#) como sugerencia de presentación.
3. Para el segundo programa use la función creada y tome como entrada una imagen de la cámara web (o vídeo en caso de no contar con cámara) y muestre solo el resultado final sobre esta imagen.
4. Presente los resultados en un archivo PDF que contenga su información personal y las imágenes solicitadas para cada sección.