| **Laboratorio Nº 2- Introducción a OpenCV** | | | |
|---|---|---|---|
| **Asignatura** | Tópicos Especiales - Visión Artificial | **Código** | 0756 |
| **Profesor** | José Carlos Rangel Ortiz | | |

# 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 solida 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

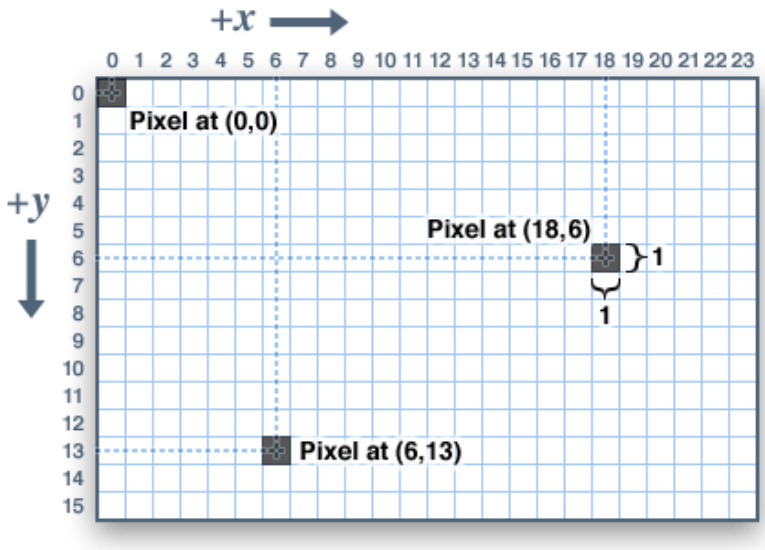



Figura 1: Origen de Coordenadas de OpenCV

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

## Objetivo

- Presentar al estudiante las operaciones básicas como transformaciones y escaldo sobre imágenes.
- Conocer las formas de aplicar filtros a imágenes utilizando OpenCV.
- Calcular histogramas de color para imágenes.

## 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. Operaciones sobre Imágenes

En esta sección realizaremos algunas transformaciones geométricas sobre imágenes de igual manera se calculará el histograma de color para una imagen.

## 1.1. Cálculo del Histograma

Para el cálculo del histograma puede usar cualquier imagen de su propiedad. El código de esta parte solo se utilizará para calcular y mostrar el histograma.

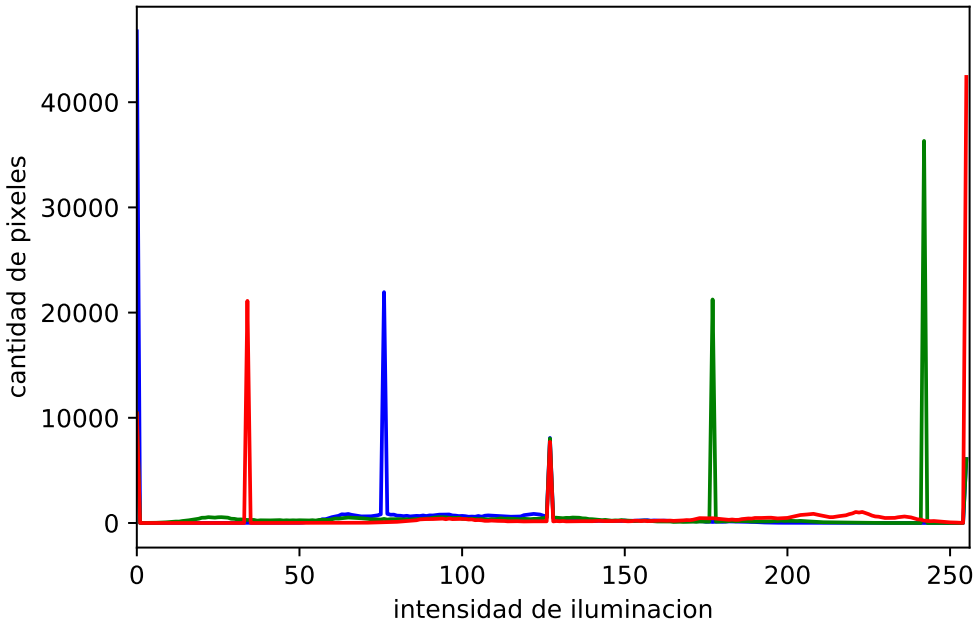
1. Cree un nuevo script e importe en este script OpenCV y MatPlotLib.
2. Cargue una imagen desde sus archivos
3. Para el cálculo del histograma se utiliza la función `cv2.hist()` como muestra el siguiente código:

```
# Arreglo para guía de los colores
colors = ('b','g','r')

for i, c in enumerate(colors):
    hist = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(hist, color = c)
    plt.xlim([0,256])
```

4. La función `cv2.hist()` recibe, la imagen, el canal, una máscara (omitida en este ejemplo), el tamaño del histograma y el rango de valores. Usando estos datos calcula la frecuencia de cada color en la imagen, para más información consulte este enlace.
5. En este fragmento se utiliza también la función `plt.plot()` para realizar la gráfica de los histogramas de cada color.
6. Como siguiente paso se configurara el plot, se guardará en memoria y se mostrará el resultado, con el siguiente código.

```
# Configuración de los Ejes
plt.xlabel('intensidad de iluminación')
plt.ylabel('cantidad de píxeles')

# Guardar el plot en formato pdf utilizando MatPlotLib
plt.savefig('out/TowerHistograma.pdf', dpi=600, bbox_inches='tight')
# Mostrar el Histograma
plt.show()
```

7. Lo cual generaría el siguiente resultado, dependiendo de la imagen elegida




## 1.2. Transformaciones Geométricas

En esta sección se recomienda utilizar un script o archivo nuevo para realizar las diferentes pruebas. En este caso se estará utilizando como imagen de prueba un archivo suministrado por el docente cuyo nombre es *lena_image.png*, el cual se encuentra entre los archivos del laboratorio.

Cuando hablamos de transformaciones geométricas en las imágenes se habla de escalas, rotaciones y traslaciones, estas tienen cada una su forma de aplicarse en OpenCV

1. Cree un nuevo script e importe en este script OpenCV, MatPlotLib y NumPy.
2. En este caso para facilitar la visualización crearemos una función al inicio de nuestro script, la cual mostrará los resultados utilizando MatPlotLib y haciendo las conversiones de color de manera directa. El código de dicha función es el siguiente:

```
def show_with_matplotlib(img, title):
    """Mostrar las imágenes usando las capacidades de MatPlotLib"""

    # Convertir imagen BGR a RGB
    img_RGB = img[:, :, ::-1]

    # Mostrar la imagen con matplotlib:
    plt.imshow(img_RGB)
    plt.title(title)
    plt.show()
```

3. Lea la imagen suministrada y almacene bajo el nombre `image`

4. Este código puede ser utilizado para todas las modificaciones que se presentaran a continuación.

### 1.2.1. Escalado

1. Escalar una imagen consiste en cambiar su tamaño, como se realizó en laboratorio anterior.
2. Para esto se utiliza el siguiente código:

```
# El parámetro interpolación especifica como se calcularan los valores
# de los píxeles en la nueva imagen
# Escalar 0.5 del valor de la imagen original
dst_image = cv2.resize(image, None,
       fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)

# Obtener las dimensiones de la imagen
height, width = image.shape[:2]

# Aplicación de un nuevo escalado basados en la imagen original
dst_image_2 = cv2.resize(image, (width * 2, height * 2),
                         interpolation=cv2.INTER_LINEAR)

# Mostrar las imagenes escaladas
show_with_matplotlib(dst_image, 'Resized image')
show_with_matplotlib(dst_image_2, 'Resized image 2')
```

3. En este caso se crea una imagen con un tamaño menor a la original y una de un tamaño mayor.

### 1.2.2. Traslaciones

1. Las traslaciones en una imagen consisten en mover los píxeles de una imagen hacia un punto definido. Para esta modificación se hace necesaria una matriz de transformaciones. Estas matrices son un elemento matemático de tamaño $2x3$ para $2D$ donde la ultima columna representa la cantidad o distancia que se moverán los elementos en cada eje de coordenadas, se define un valor para $\widehat{x}$ y otro para $\widehat{y}$. Esta matriz se crea con el siguiente código:

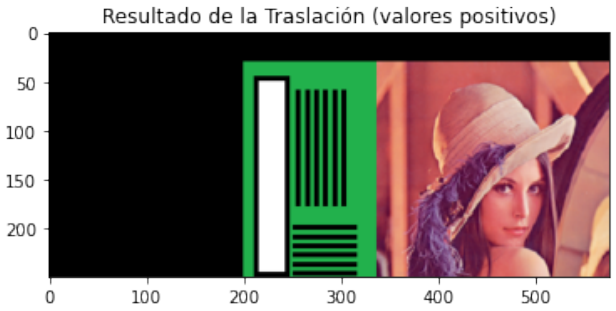
```
# Matriz de Transformación
M = np.float32([[1, 0, 200], [0, 1, 30]])
```

2. La traslación se aplica entonces usando la función `cv2.warpAffine()` como se muestra en el siguiente código:

```
# Esta matriz se pasa a la función cv2.warpAffine():
dst_image = cv2.warpAffine(image, M, (width, height))
```

```
# Mostrar el resultado
show_with_matplotlib(dst_image,
           'Resultado de la Traslación (valores positivos)')
```

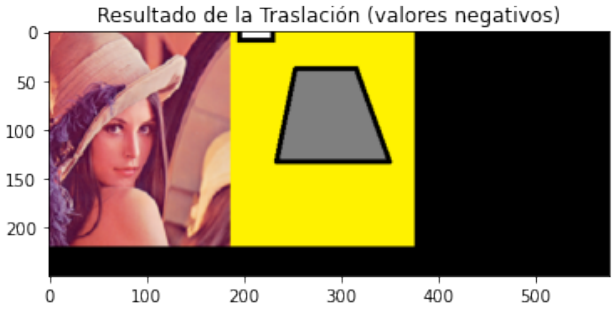
3. Lo cual generaría el siguiente resultado




4. De igual manera se pueden aplicar valores Negativos en la matriz como se muestra a continuación

```
# La matriz también puede contener valores negativos
M = np.float32([[1, 0, -200], [0, 1, -30]])
dst_image = cv2.warpAffine(image, M, (width, height))

# Mostrar el resultado
show_with_matplotlib(dst_image,
       'Resultado de la Traslación (valores negativos)')
```

5. Lo cual generaría el siguiente resultado




### 1.2.3. Rotaciones

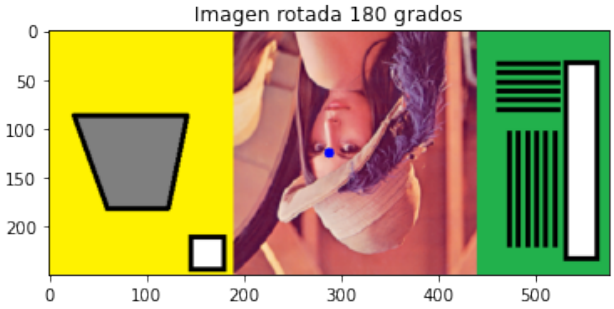
1. En el caso de las rotaciones se procede de manera similar a las traslaciones, en este caso la matriz de transformaciones contendrá los datos de la rotación a seguir en los $2$ primeros elementos de cada fila de la matriz.

2. Para crear una matriz de transformación con información de rotación se utiliza la función `cv2.getRotationMatrix2D`, la cual recibe las coordenadas del punto central de la imagen y el ángulo deseado. Usando esto calcula cual sería la matriz para que una imagen gire el ángulo deseado.

3. El siguiente código crea la matriz de transformación, y la aplica en la imagen. Se hace el procedimiento para $180^o$

```
# Creación de Matriz de Rotación para 180 grados
# getRotationMatrix2D(Point2f center, double angle, double scale)
M = cv2.getRotationMatrix2D((width / 2.0, height / 2.0), 180, 1)
dst_image = cv2.warpAffine(image, M, (width, height))

# Mostrar el centro de rotación
cv2.circle(dst_image, (round(width / 2.0), round(height / 2.0)),
                                          5, (255, 0, 0), -1)
show_with_matplotlib(dst_image, 'Imagen rotada 180 grados')
```

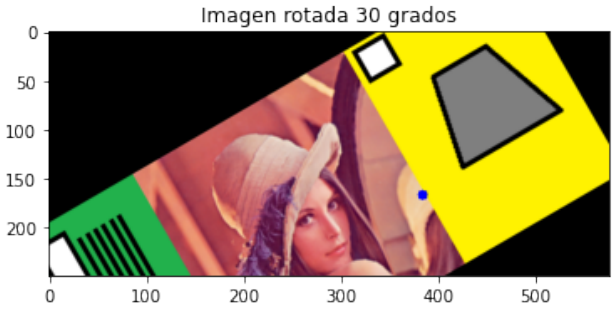
4. Lo cual generaría el siguiente resultado para $180^o$




5. Se puede también cambiar el punto pivote o centro de rotación, en este caso se realiza para un giro de $30$

```
# En este caso cambiamos el centro de rotación
M = cv2.getRotationMatrix2D((width / 1.5, height / 1.5), 30, 1)
dst_image = cv2.warpAffine(image, M, (width, height))

# Mostrar el centro de rotación y rotar 30 grados
cv2.circle(dst_image, (round(width / 1.5), round(height / 1.5)),
                                          5, (255, 0, 0), -1)
show_with_matplotlib(dst_image, 'Imagen rotada 30 grados')
```

6. Lo cual generaría el siguiente resultado para $30^o$




### 1.2.4. Transformaciones Afines

En una transformación afín los puntos, líneas rectas y planos son preservados. Las líneas paralelas siguen siendo paralelas, pero no se conservan los ángulos y distancias entre los puntos.

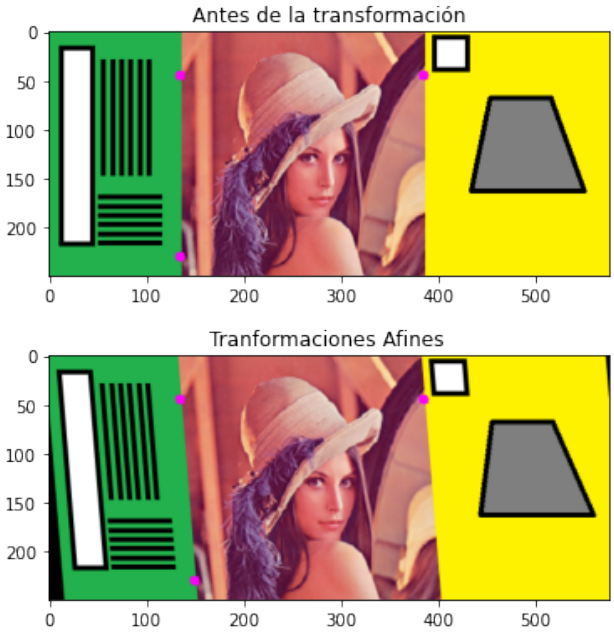
1. En esta utilizamos la función `cv2.getAffineTransform()` para construir la matriz de transformación se definen un conjunto de puntos de referencia en la imagen original y se define cual es la posición deseada de esos puntos en la imagen transformada.

2. Siguiendo con el código de la sección anterior, copie lo siguiente en su archivo.

```
# En una copia de la imagen se dibujan los puntos de referencia
image_points = image.copy()
cv2.circle(image_points, (135, 45), 5, (255, 0, 255), -1)
cv2.circle(image_points, (385, 45), 5, (255, 0, 255), -1)
cv2.circle(image_points, (135, 230), 5, (255, 0, 255), -1)

# Mostrar los puntos de referencia
show_with_matplotlib(image_points, 'Antes de la transformación')

# Se crean 2 arreglos con los puntos de referencia
# y su ubicación deseada.
pts_1 = np.float32([[135, 45], [385, 45], [135, 230]])
pts_2 = np.float32([[135, 45], [385, 45], [150, 230]])

# Usando getAffineTransform se obtiene la matriz de transformación
M = cv2.getAffineTransform(pts_1, pts_2)
dst_image = cv2.warpAffine(image_points, M, (width, height))

# Mostrar el resultado:
show_with_matplotlib(dst_image, 'Tranformaciones Afines ')
```

3. En este código la función `cv2.getAffineTransform` se encarga de calcular cual es la transformación que se deben aplicar a la imagen para que los puntos de referencia se ubiquen en la posición deseada.

4. Lo cual generaría el siguiente resultado




### 1.2.5. Cropping

El cropping consiste en recortar una parte de la imagen original, para ello se debe definir el área que deseamos cortar.

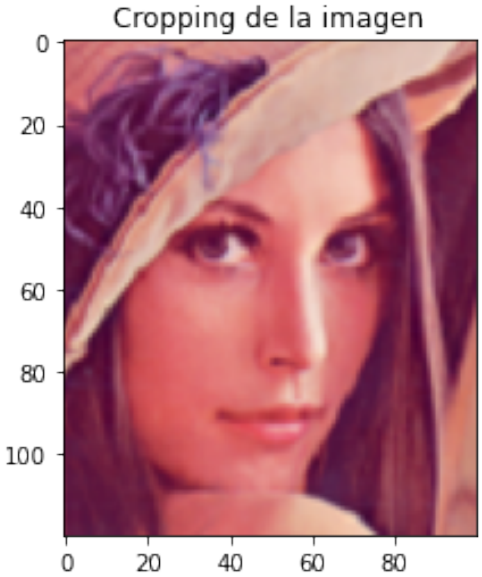
1. Utilizando la imagen del código anterior, definimos una zona o parche de la misma manera que se realizó en el laboratorio anterior.

2. Copiamos el siguiente código:

```
# Se crea en una copia la imagen para mostrar los puntos
# que forman parte del proceso
image_points = image.copy()

# Mostrar puntos y las líneas que conectan dichos puntos
cv2.circle(image_points, (230, 80), 5, (0, 0, 255), -1)
cv2.circle(image_points, (330, 80), 5, (0, 0, 255), -1)
cv2.circle(image_points, (230, 200), 5, (0, 0, 255), -1)
cv2.circle(image_points, (330, 200), 5, (0, 0, 255), -1)
cv2.line(image_points, (230, 80), (330, 80), (0, 0, 255))
cv2.line(image_points, (230, 200), (330, 200), (0, 0, 255))
cv2.line(image_points, (230, 80), (230, 200), (0, 0, 255))
cv2.line(image_points, (330, 200), (330, 80), (0, 0, 255))

# Mostrar la imagen
```

```
show_with_matplotlib(image_points, 'Antes del cropping')

# Para el recorte e utiliza la función slicing de numpy:
dst_image = image[80:200, 230:330]

# Mostrar la imagen
show_with_matplotlib(dst_image, 'Cropping de la imagen')
```

3. 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. Aplique para una imagen de su elección (diferente a la utilizada en el ejemplo), un cropping del elemento u objeto principal de dicha imagen.
2. Utilizando el recorte anterior aplique una rotación de $167$ grados utilizando como pivote un punto desplazado $45$ píxeles del centro en el eje $\widehat{x}$ y para el eje $\widehat{y}$ utilice el centro del eje.
3. En otra copia del recorte realice una traslación de $-55$ píxeles en $\widehat{x}$ y $89$ píxeles en el eje $\widehat{y}$.
4. En este momento debe tener 4 archivos de imagen en memoria (original, crop, rotada y trasladada), calcule el histograma de color para cada una de estas imágenes.
5. Adjunte el resultado de cada modificación y el histograma generado para cada imagen.

# 2. Aplicación de Filtros

En esta sección se trataran los procedimientos para la modificación de imágenes utilizando filtros, el cual es uno de los principales usos de OpenCV. Los filtros nos permiten resaltar características y atributos en las imágenes lo cual permite apreciar mejor dichos elementos y también facilita el procesamiento de algunas imágenes.

Un filtro se pude entender como una matriz de tamaño reducido, en cada elemento de esta matriz se almacena un valor numérico. Cuando se aplica un filtro a una imagen ocurre una operación matemática entre los valores del filtro y los valores de cada canal de la imagen.

Existen diferentes tipos de filtros, enfocados en diferentes aspectos, de igual manera se pueden definir filtros propios para aplicarlos a nuestras imágenes.

Muchos filtros o también conocidos como *kernels* se enfocan en eliminar el ruido en la imagen, otros en desenfocar el contenido de esta, otros se encargan de resaltar los bordes o contornos de los elementos de la imagen.

## 2.1. Filtros Arbitrarios

El código que implemente en esta parte le funcionará para los siguientes pasos de esta sección del laboratorio.

1. En este caso se utilizan filtros definidos a mano por el programador. OpenCV cuenta con la función `cv2.filter2D()` la cual se encarga de aplicar un filtro arbitrario, definido en una matriz a una imagen elegida por el usuario.

2. Como primer paso se debe crear un *script* nuevo e importar en este OpenCV, NumPy y MatPlotLib.

3. Este *script* utilizará una versión modificada de la función `show_with_matplotlib` que se utilizó en la sección previa. Agregue el siguiente código al su archivo:

```
def show_with_matplotlib(color_img, title, pos):
    """Mostar una imagen usando MatPlotLib"""

    # Convertir imagen BGR a RGB
    img_RGB = color_img[:, :, ::-1]

    ax = plt.subplot(3, 3, pos)
    plt.imshow(img_RGB)
    plt.title(title)
    plt.axis('off')
```

4. Cargue una imagen de su preferencia, en el ejemplo el nombre de la variable para la imagen será `image` y a continuación en el *script*, copie el siguiente código:

```
# crear un objeto figura con las dimensiones adecuadas
plt.figure(figsize=(12, 6))
plt.suptitle("Técnicas de Smoothing", fontsize=14, fontweight='bold')

# Creamos el kernel para el suavizado de la imagen
# En este caso se crea un kernel (10,10)
kernel_averaging_10_10 = np.ones((10, 10), np.float32) / 100


# kernel_averaging_5_5 = np.ones((5, 5), np.float32)/25
kernel_averaging_5_5 = np.array([[0.04, 0.04, 0.04, 0.04, 0.04],
                                 [0.04, 0.04, 0.04, 0.04, 0.04],
                                 [0.04, 0.04, 0.04, 0.04, 0.04],
                                 [0.04, 0.04, 0.04, 0.04, 0.04],
                                 [0.04, 0.04, 0.04, 0.04, 0.04]])

print("kernel: {}".format(kernel_averaging_5_5))

# La funcioón cv2.filter2D() aplica el filtro linea arbitrario
smooth_image_f2D_5_5 = cv2.filter2D(image, -1, kernel_averaging_5_5)
smooth_image_f2D_10_10 = cv2.filter2D(image, -1, kernel_averaging_10_10)
```

5. En este ejemplo se puede ver como se definen 2 filtros arbitrarios y son aplicados a la imagen usada en el estudio.

6. Este código prepara una figura donde se mostraran todos los filtros que se aplicaran en el laboratorio.

## 2.2. Aplicación de Filtros de OpenCV

En esta parte continuaremos con el script anterior. En este caso aplicaremos a la imagen algunos filtros que están disponibles en la librería. Se mostrará el código de cada uno y al final de esta sección se mostrarán los resultados obtenidos.

1. Filtro de suavizado mediante promediado

```
# La funcion cv2.blur() suaviza una imagen
smooth_image_b = cv2.blur(image, (10, 10))
```

2. Suavizado mediante el `boxFilter`

```
# Filtro de Suavizado
smooth_image_bfi = cv2.boxFilter(image, -1, (10, 10), normalize=True)
```

3. La función `cv2.GaussianBlur()` aplica una convolución en la imagen. Este kernel se puede controlar usando los parámetros *(kernel size)*, la desviación estándar y la dirección del kernel Gaussiano.

```
# Filtro Gaussiano
smooth_image_gb = cv2.GaussianBlur(image, (9, 9), 0)
```

4. Para difuminar una imagen se puede aplicar `cv2.medianBlur()`

```
# Filtro de difuminación de imágenes
smooth_image_mb = cv2.medianBlur(image, 9)
```

5. Para la reducción del ruido se puede aplicar un filtro bilateral con la función `cv2.bilateralFilter()`

```
# Reducción de ruido
smooth_image_bf = cv2.bilateralFilter(image, 5, 10, 10)
smooth_image_bf_2 = cv2.bilateralFilter(image, 9, 200, 200)
```

## 2.3. Graficando los resultados

1. En este paso procederemos a colocar cada una de las imágenes, a las cuales se les ha aplicado un filtro, en una sola ventana de visualización, utilizando las capacidades de MatPlotLib, recordemos que al inicio de esta sección se configuró la figura que contendrá todos los resultados.

2. Para mostrar la figura, se debe copiar el siguiente fragmento

```
# Graficar las images:
show_with_matplotlib(image, "original", 1)
show_with_matplotlib(smooth_image_f2D_5_5, "cv2.filter2D() (5,5) kernel", 2)
show_with_matplotlib(smooth_image_f2D_10_10, "cv2.filter2D() (10,10) kernel", 3)
show_with_matplotlib(smooth_image_b, "cv2.blur()", 4)
show_with_matplotlib(smooth_image_bfi, "cv2.boxFilter()", 5)
show_with_matplotlib(smooth_image_gb, "cv2.GaussianBlur()", 6)
show_with_matplotlib(smooth_image_mb, "cv2.medianBlur()", 7)
show_with_matplotlib(smooth_image_bf, "cv2.bilateralFilter() - small values", 8)
show_with_matplotlib(smooth_image_bf_2, "cv2.bilateralFilter() - big values", 9)

# Mostrar los resultados de la aplicación en lugar de un alto mando
plt.show()
```

3. Lo cual generaría el siguiente resultado, si esta ejecutando en Jupyter el formato de la imagen resultante variará un poco.


## 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. Aplique para una imagen de su elección un filtro arbitrario de tamaño $7x7$ y cuyo valor en cada elemento sea $\frac{1}{43}$.
2. Aplique para la misma imagen un filtro arbitrario de tamaño $3x3$ y cuyo valor en cada elemento sea $\frac{1}{6}$.
3. Seleccione $2$ de los filtros mostrados en el ejemplo y aplíquelos en su imagen $3$ veces utilizando parámetros diferentes para el filtro en cada aplicación
4. Muestre en una sola ventana la imagen original más el resultado de aplicar los filtros, esta gráfica debe contener $9$ elementos.