



## Tratamiento de imágenes: 3

**Hugo Coll Ferri**

**hucolfer@dcom.upv.es**



Tratamiento de imagen digital

1



1. Funciones de dibujo
2. Mejora del contraste de una imagen
3. Detección de bordes
4. Desenfoque de movimiento (Blurring/Motion Blur)
5. Mejora de una imagen (Sharpening)
6. Erosión y dilatación
7. Creación de filtro de viñeta
8. Creación de filtro de relieve



Tratamiento de imagen digital

2




UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA




1. Funciones de dibujo



Tratamiento de imagen digital



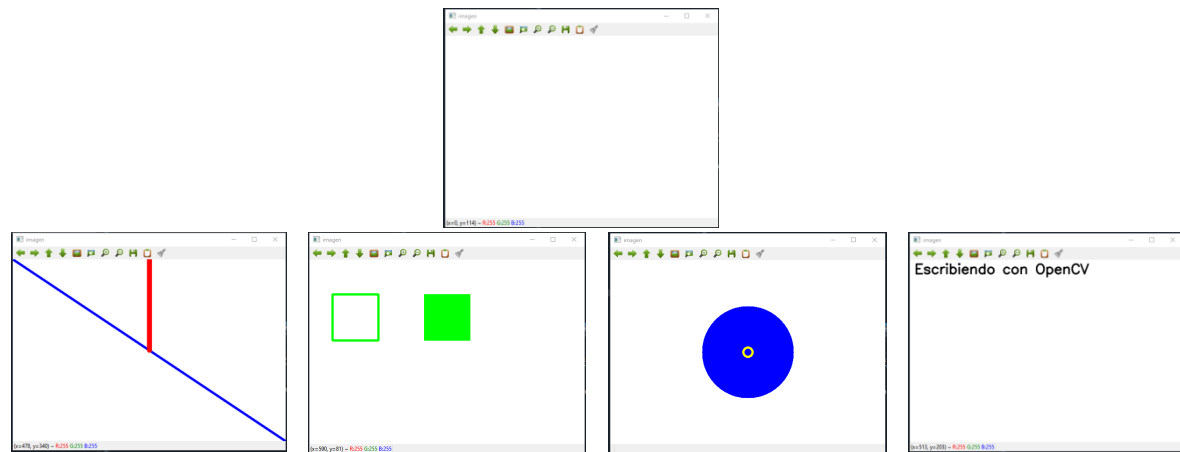
UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA




1. Funciones de dibujo

En muchas de las aplicaciones de visión por computador son dibujados círculos, líneas, rectángulos o texto.

Vamos a estudiar las funciones necesarias para dibujar cada una de ellas en una imagen.





Tratamiento de imagen digital



## 1. Funciones de dibujo



### Objetivo: Fondo de la imagen

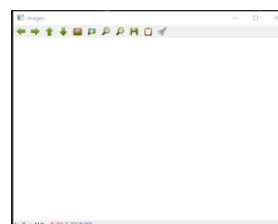
Antes de comenzar a dibujar deberemos tener un fondo, que será el lienzo sobre el se representarán las imágenes.

- En la **línea 1 y 2** importamos OpenCV y numpy con un alias **np**.
- En la **línea 4** construimos una imagen con ayuda de **np.ones**, en donde construiremos una matriz de unos, de 400 filas por 600 columnas y 3 canales, lo que nos ayudará a crear una imagen para manejar los canales B, G y R, seguido de **dtype=np.uint8**. Esta matriz es multiplicada por 255, con ello cada uno de los canales adquirirá dicho valor, entonces la imagen se mostrará en blanco totalmente. ¿Recuerdas que (255, 255, 255) es la representación de blanco en BGR?.
- En las **líneas 6 a 8** visualizamos la imagen, establecemos que la visualización se detenga hasta que sea presionada una tecla con **cv2.waitKey(0)** y que finalmente se cierren las ventanas con **cv2.destroyAllWindows()**

```
import cv2
import numpy as np

imagen = 255*np.ones((400,600,3),dtype=np.uint8)

cv2.imshow('imagen',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Tratamiento de imagen digital



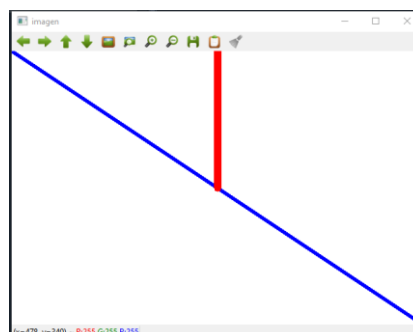
## 1. Funciones de dibujo



### Objetivo: Dibujar líneas

Para dibujar una línea en OpenCV, vamos a emplear la función **cv2.line**, en ella debemos especificar:

- La imagen en donde se va a visualizar.
- Las coordenadas del punto inicial en x e y.
- Las coordenadas del punto final en x e y.
- El color en BGR.
- Grosor de línea.



Tratamiento de imagen digital



# 1. Funciones de dibujo



## Objetivo: Dibujar líneas

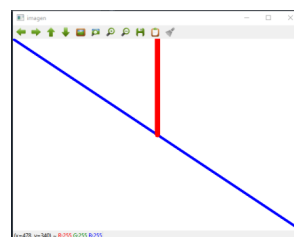
- **Línea 7:** Usamos la función **cv2.line**, en ella especificamos que se va a dibujar en la variable **imagen**, el punto inicial está en **(0,0)** en x e y, mientras que el punto final está en **(600,400)**, el color en BGR es **(255,0,0)**, que corresponde al color azul, mientras que el grosor de línea es de 4 píxeles.
- **Línea 8:** Esta es similar a la anterior, las diferencias están en los datos ingresado, pues el punto inicial está en **(300,0)**, mientras que el final en **(300,200)**, se visualizará en un color rojo **(255,100,255)**, mientras que el grosor de línea será de 10.

```
import cv2
import numpy as np

imagen = 255*np.ones((400,600,3),dtype=np.uint8)

#Dibujando lineas
cv2.line(imagen,(0,0),(600,400),(255,0,0),4)
cv2.line(imagen,(300,0),(300,200),(0,0,255),10)

cv2.imshow('imagen',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Tratamiento de imagen digital

7



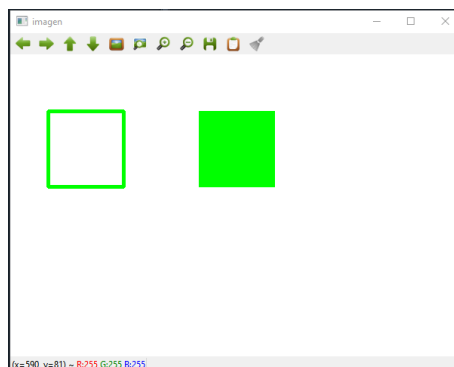
# 1. Funciones de dibujo



## Objetivo: Dibujar rectángulos

Para dibujar rectángulos en OpenCV, vamos a emplear la función **cv2.rectangle**, en ella debemos especificar:

- La imagen en donde se va a visualizar.
- Las coordenadas del punto inicial en x e y.
- Las coordenadas del punto final en x e y.
- El color en BGR.
- Grosor de línea.



Tratamiento de imagen digital

8



## 1. Funciones de dibujo



### Objetivo: Dibujar rectángulos

**Línea 7:** Usamos la función `cv2.rectangle`, en ella especificamos que el rectángulo se va a dibujar en la variable **imagen**, el punto inicial está en **(50,75)**, x e y, mientras que el inferior derecho está en **(150,175)**, el color en BGR es **(0,255,0)** que corresponde al color verde, mientras que el grosor de línea es de 3 pixels.

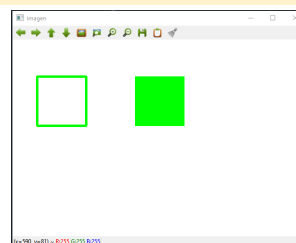
**Línea 8:** Los valores de los argumentos ingresado son similares al de la línea anterior, pero el grosor de línea es un número negativo. Si damos un número negativo en el grosor de línea, este va a hacer que el rectángulo se dibuje totalmente lleno.

```
import cv2
import numpy as np

imagen = 255*np.ones((400,600,3),dtype=np.uint8)

#Dibujando un rectangulo
cv2.rectangle(imagen,(50,75),(150,175),(0,255,0),3)
cv2.rectangle(imagen,(250,75),(350,175),(0,255,0),-1)

cv2.imshow('imagen',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Tratamiento de imagen digital



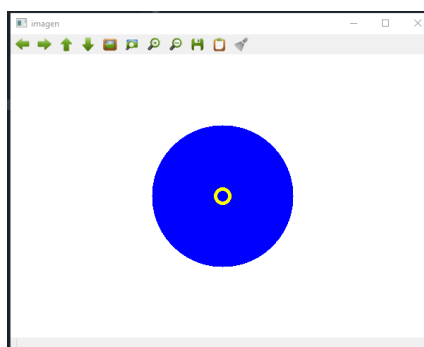
## 1. Funciones de dibujo



### Objetivo: Dibujar círculos

Para poder dibujar círculos en OpenCV, vamos a emplear la función `cv2.circle`, en ella debemos especificar:

- La imagen en donde se va a visualizar.
- Las coordenadas del punto central.
- Radio del círculo a dibujar.
- El color en BGR.
- Grosor de línea.



Tratamiento de imagen digital



## 1. Funciones de dibujo



### Objetivo: Dibujar círculos

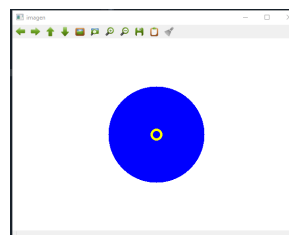
- **Línea 7:** Se dibujará un círculo con **cv2.circle**, la imagen en la que se visualizará es en **imagen**, el punto central del círculo está en **(300,200)**, con un radio de **100** y color azul con **(255,0,0)**. Mientras que el grosor de línea es **-1**, un número negativo, por ello al igual que con los rectángulos, el círculo se llenará.
- **Línea 8:** Ahora en esta línea especificamos que el círculo se va a dibujar en la variable **imagen**, el punto central está en **(300,200)**, x e y, con un radio de **10**, el color en BGR es **(20,255,255)** que corresponde al color amarillo, mientras que el grosor de línea es de 3 píxeles.

```
import cv2
import numpy as np

imagen = 255*np.ones((400,600,3),dtype=np.uint8)

#Dibujando círculos
cv2.circle(imagen,(300,200),100,(255,0,0),-1)
cv2.circle(imagen,(300,200),10,(0,255,255),3)

cv2.imshow('imagen',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Tratamiento de imagen digital



## 1. Funciones de dibujo



### Objetivo: Dibujar texto

Para poder visualizar texto en OpenCV, vamos a emplear la función **cv2.putText**, en ella debemos especificar:

- La imagen en donde se va a visualizar.
- El texto que se va a visualizar.
- Ubicación de la esquina inferior izquierda del texto en la imagen.
- Fuente del texto.
- Tamaño del texto.
- El color en BGR.
- Grosor de línea para el texto.



Tratamiento de imagen digital



## 1. Funciones de dibujo

### Objetivo: Dibujar texto

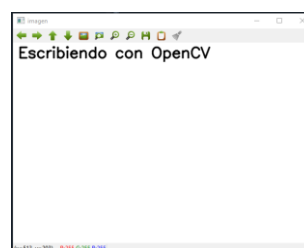
- En la **línea 6** se especifica el tipo de fuente del texto que se va a visualizar en la imagen, esta se va a utilizar a continuación, en la **línea 7**.
- Línea 7:** Se usa **cv2.putText**, para poder ubicar texto en la imagen. Se presenta la imagen (**imagen**) en donde se va a visualizar que sería el texto '**Practicando con OpenCV**', su ubicación en las coordenadas x e y (**10,30**), la fuente que está almacenada en **font**, el tamaño que sería **1**, el color negro (**0,0,0**), con un grosor de línea de **2**.

```
import cv2
import numpy as np

imagen = 255*np.ones((400,600,3),dtype=np.uint8)

font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(imagen,'Escribiendo con OpenCV',(10,30),font,1,(0,0,0),2,cv2.LINE_AA)

cv2.imshow('imagen',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Tratamiento de imagen digital



## 1. Funciones de dibujo

### Objetivo: Dibujar texto

Varias fuentes simples están disponibles en OpenCV que se pueden usar para escribir texto.

- FONT\_HERSHEY\_SIMPLEX = 0
- FONT\_HERSHEY\_PLAIN = 1
- FONT\_HERSHEY\_DUPLEX = 2
- FONT\_HERSHEY\_COMPLEX = 3
- FONT\_HERSHEY\_TRIPLEX = 4
- FONT\_HERSHEY\_COMPLEX\_SMALL = 5
- FONT\_HERSHEY\_SCRIPT\_SIMPLEX = 6
- FONT\_HERSHEY\_SCRIPT\_COMPLEX = 7

Se puede indicar la fuente por su **número de fuente asignado**:

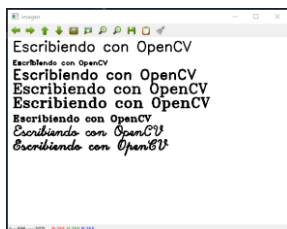
```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(imagen,'Escribiendo con OpenCV',(10,30),Num. fuente,1,(0,0,0),2,cv2.LINE_AA)
```

Tratamiento de imagen digital



## 1. Funciones de dibujo

**Objetivo: Dibujar texto**



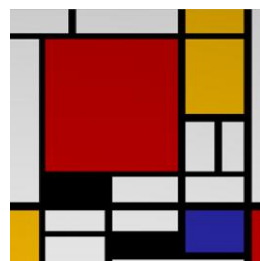
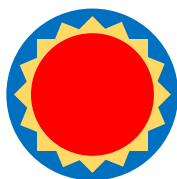
```
import cv2
import numpy as np

imagen = 255*np.ones((400,600,3),dtype=np.uint8)

cv2.putText(imagen,'Escribiendo con OpenCV',(10,30),0,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,60),1,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,90),2,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,120),3,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,150),4,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,180),5,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,210),6,1,(0,0,0),2)
cv2.putText(imagen,'Escribiendo con OpenCV',(10,240),7,1,(0,0,0),2)
cv2.imshow('imagen',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



**Ejercicios. Dibujar las siguientes imágenes**







## 2. Mejora del contraste de una imagen



## 2. Mejora del contraste de una imagen

Cuando capturamos imágenes en condiciones de poca luz, las imágenes resultan oscuras.

Muchos detalles de la imagen no son claramente visibles para el ojo humano.

Al ojo humano le gusta el contraste, por lo que debemos ajustar el contraste para que la imagen se vea agradable y agradable.

Mediante el proceso de ecualización de histogramas se puede lograr esto.





## 2. Mejora del contraste de una imagen



### Ecualización de imagen en B/N

Si la imagen de entrada es muy oscura → Podemos corregirla.

- Necesitamos ajustar los valores de los píxeles para que se distribuyan en todo el espectro de valores, es decir, entre 0 y 255.

```
import cv2
import numpy as np

img = cv2.imread('input.jpg', 0)

# Ecualizar el histograma de la imagen de entrada
histeq = cv2.equalizeHist(img)

cv2.imshow('Input', img)
cv2.imshow('Histogram equalized', histeq)
cv2.waitKey(0)
```



## 2. Mejora del contraste de una imagen



### Ecualización de imagen en color

La ecualización de histogramas es un proceso no lineal. Por lo tanto, no podemos simplemente separar los tres canales en una imagen RGB, igualar el histograma por separado y combinarlos más tarde para formar la imagen de salida.

El concepto de ecualización de histograma solo es aplicable a los valores de intensidad en la imagen. Por lo tanto, debemos asegurarnos de no modificar la información de color cuando hagamos esto.

Para manejar la ecualización del histograma de las imágenes en color, debemos convertirlo en un espacio de color donde la intensidad se separe de la información del color.

YUV es un buen ejemplo de este tipo de espacio de color.

Una vez que lo convertimos a YUV, solo necesitamos ecualizar el canal Y y combinarlo con los otros dos canales para obtener la imagen de salida.





UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

## 2. Mejora del contraste de una imagen



### Ecualización de imagen en color

Se aplicará el siguiente script:

```
import cv2
import numpy as np

img = cv2.imread('input.jpg.jpg')

img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

# Ecualiza el histograma del canal Y
img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])

# Convierte la imagen YUV de nuevo a formato RGB
img_output = cv2.cvtColor(img_yuv,
cv2.COLOR_YUV2BGR)

cv2.imshow('Color input image', img)
cv2.imshow('Histogram equalized', img_output)

cv2.waitKey(0)
```

Tratamiento de imagen digital



21



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



## 3. Detección de bordes



Tratamiento de imagen digital

22

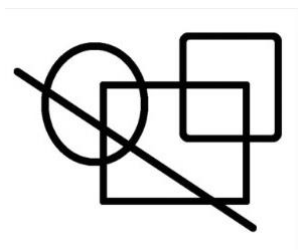


### 3. Detección de bordes



El proceso de detección de bordes implica detectar bordes afilados en la imagen y producir una imagen binaria como salida.

Por lo general, se dibujan líneas blancas sobre un fondo negro para indicar esos bordes.



### 3. Detección de bordes



Un **filtro de detección de bordes simple** es el conocido como filtro **Sobel**.

Dado que los bordes pueden ocurrir tanto en dirección horizontal como vertical, el filtro Sobel se compone de los siguientes dos núcleos:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

El núcleo de la izquierda detecta los bordes horizontales y el núcleo de la derecha detecta los bordes verticales.



### 3. Detección de bordes

Código para usar **filtros Sobel** para detectar bordes:

```
import cv2

# Cargamos la imagen en formato BGR
img = cv2.imread(imagen.jpg')

# Convertimos la imagen a escala de grises
img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Guardamos la imagen en escala de grises en un archivo nuevo
cv2.imwrite("imagen_gris.jpg", img_gris)

rows, cols = img_gris.shape

sobel_horizontal = cv2.Sobel(img_gris, cv2.CV_64F, 1, 0, ksize=5)
sobel_vertical = cv2.Sobel(img_gris, cv2.CV_64F, 0, 1, ksize=5)

cv2.imshow('Original', img)
cv2.imshow('Sobel horizontal', sobel_horizontal)
cv2.imshow('Sobel vertical', sobel_vertical)

cv2.waitKey(0)
```

Tratamiento de imagen digital

25

### 3. Detección de bordes

También se puede usar el **filtro Laplaciano**.

La ventaja de usar este filtro es que usa doble derivada en ambas direcciones.

Su principal problema es que da una **salida ruidosa**.

Para mejorarlo se usa el **filtro Canny**.

```
import cv2

# Cargamos la imagen en formato BGR
img = cv2.imread(imagen.jpg')

# Convertimos la imagen a escala de grises
img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Guardamos la imagen en escala de grises en un archivo nuevo
cv2.imwrite("imagen_gris.jpg", img_gris)

rows, cols = img_gris.shape

laplacian = cv2.Laplacian(img, cv2.CV_64F)

cv2.imshow('Original', img)
cv2.imshow('laplacian', laplacian)

cv2.waitKey(0)
```

Tratamiento de imagen digital

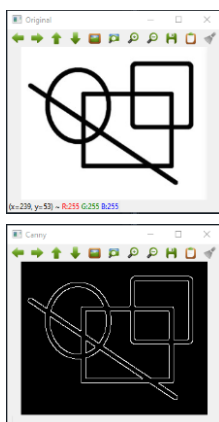
26



### 3. Detección de bordes



Código para usar **filtro Canny**:



```
import cv2

# Cargamos la imagen en formato BGR
img = cv2.imread('imagen.jpg')

# Convertimos la imagen a escala de grises
img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Guardamos la imagen en escala de grises en un archivo nuevo
cv2.imwrite("imagen_gris.jpg", img_gris)

rows, cols = img_gris.shape

canny = cv2.Canny(img, 50, 240)

cv2.imshow('Original', img)
cv2.imshow('Canny', canny)

cv2.waitKey(0)
```



Tratamiento de imagen digital

27



### 4. Desenfoque de movimiento



Tratamiento de imagen digital

28



## 4. Desenfoque de movimiento



Si aplicamos el efecto de desenfoque de movimiento (Blurring/Motion Blur) , parecerá que capturamos la imagen mientras nos movíamos en una dirección particular.

Por ejemplo, se puede hacer que una imagen se vea como si hubiera sido capturada desde un automóvil en movimiento.



Tratamiento de imagen digital

29



## 4. Desenfoque de movimiento



Lo generamos con el script:

La función **cv2.filter2D** tiene los siguientes parámetros:

- **imagen**: es la imagen de entrada a la que se aplicará el filtro de convolución. Puede ser una imagen en escala de grises o una imagen en color.
- **-1**: es el tipo de profundidad de bits de la imagen de salida. En este caso, se utiliza -1 para que la imagen de salida tenga la misma profundidad de bits que la imagen de entrada.
- **kernel**: es el kernel que se utilizará para la convolución. El kernel es una matriz 2D que especifica cómo se realizará la convolución. La forma y los valores de los elementos en el kernel determinan el tipo de efecto que se aplicará a la imagen de entrada.

```
import cv2
import numpy as np

img = cv2.imread('input.jpg')
cv2.imshow('Original', img)

size = 15

# Generación del kernel
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size-1)/2), :] = np.ones(size)
kernel_motion_blur = kernel_motion_blur / size

# Aplicación del Kernel a la imagen de entrada
output = cv2.filter2D(img, -1, kernel_motion_blur)

cv2.imshow('Motion Blur', output)
cv2.waitKey(0)
```



Tratamiento de imagen digital

30



## 5. Mejora de una imagen (Sharpening)



Tratamiento de imagen digital

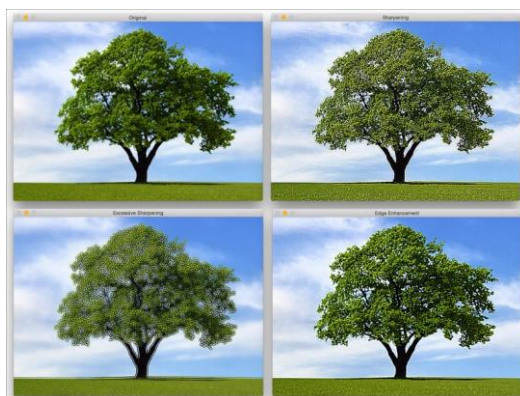
31



## 5. Mejora de una imagen (Sharpening)

Para mejorar la calidad de una imagen podemos aplicar el filtro de nitidez (**sharpening**), que agudizará los bordes de la imagen.

Este filtro es muy útil cuando queremos realzar los bordes de una imagen que no es nítida.



Tratamiento de imagen digital

32



## 5. Mejora de una imagen (Sharpening)

El nivel de nitidez depende del tipo de kernel que utilicemos. Aquí tenemos mucha libertad para personalizar el kernel, y cada kernel le dará un tipo diferente de nitidez.

$$M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Si queremos hacer una nitidez excesiva, como en la imagen inferior izquierda, usaríamos el siguiente kernel:

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Pero el **problema** con estos dos núcleos es que la **imagen de salida parece mejorada artificialmente**.

Si queremos que nuestras imágenes se vean más naturales, usaríamos un filtro de mejora de bordes.

Se usaría un **kernel gaussiano** aproximado para construir este filtro. Nos ayudará a suavizar la imagen cuando realcemos los bordes, haciendo así que la imagen se vea más natural.



Tratamiento de imagen digital

33

## 5. Mejora de una imagen (Sharpening)

El código para generarlos es:

```
import cv2
import numpy as np

img = cv2.imread('imagen.jpg')
cv2.imshow('Original', img)

# Generando los kernels
kernel_sharpen_1 = np.array([[[-1,-1,-1], [-1,9,-1], [-1,-1,-1]]])
kernel_sharpen_2 = np.array([[1,1,1], [1,-7,1], [1,1,1]])
kernel_sharpen_3 = np.array([[[-1,-1,-1,-1,-1],
                               [-1,2,2,2,-1],
                               [-1,2,8,2,-1],
                               [-1,2,2,2,-1],
                               [-1,-1,-1,-1,-1]]] / 8.0)

# Aplicando diferentes kernels a la imagen de entrada
output_1 = cv2.filter2D(img, -1, kernel_sharpen_1)
output_2 = cv2.filter2D(img, -1, kernel_sharpen_2)
output_3 = cv2.filter2D(img, -1, kernel_sharpen_3)

cv2.imshow('Sharpening', output_1)
cv2.imshow('Excessive Sharpening', output_2)
cv2.imshow('Edge Enhancement', output_3)
cv2.waitKey(0)
```



Tratamiento de imagen digital

34



## 6. Erosión y dilatación



## 6. Erosión y dilatación

La **erosión** y la **dilatación** son operaciones de **procesamiento de imágenes morfológicas**.

El procesamiento de imágenes morfológicas se ocupa básicamente de **modificar estructuras geométricas** en la imagen.

Estas operaciones se definen principalmente **para imágenes binarias**, pero **también** podemos usarlas en imágenes en **escala de grises**.

La **erosión** básicamente **elimina la capa más externa de píxeles** en una estructura, mientras que la **dilatación** **agrega una capa adicional** de píxeles en una estructura.





## 6. Erosión y dilatación

OpenCV proporciona funciones para erosionar y dilatar directamente una imagen.

- Se denominan erosionar (**cv2.erode**) y dilatar (**cv2.dilate**).
- Lo interesante de notar es el tercer argumento en estas dos funciones.
- El número de iteraciones determinará cuánto desea erosionar/dilatar una imagen determinada.
- Básicamente aplica la operación sucesivamente a la imagen resultante.

```
import cv2
import numpy as np

img = cv2.imread('D:\imagen\pruebatexto2.jpg', 0)

kernel = np.ones((5,5), np.uint8)

img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
```



## 7. Creación de filtro de viñeta



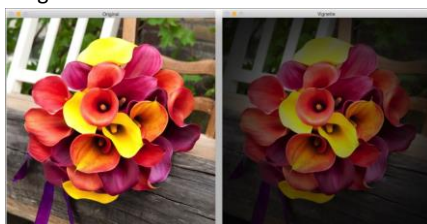
## 7. Creación de filtro de viñeta

El **filtro Viñeta** básicamente **enfoca el brillo en una parte** particular de la imagen y las **otras partes se ven desvanecidas**.

Para lograr esto, necesitamos filtrar cada canal en la imagen usando un núcleo gaussiano. OpenCV proporciona una función para hacer esto, que se llama **getGaussianKernel**.

Necesitamos construir un núcleo 2D cuyo tamaño coincida con el tamaño de la imagen.

El segundo parámetro de la función, **getGaussianKernel**, es interesante. Es la desviación estándar de la Gaussiana y controla el radio de la región central brillante.



Tratamiento de imagen digital

## 7. Creación de filtro de viñeta

- Una vez que construimos el kernel 2D, necesitamos construir una máscara normalizando este kernel y ampliándolo.  

$$\text{mask} = 255 * \text{kernel} / \text{np.linalg.norm}(\text{kernel})$$
- Una vez que construimos el kernel 2D, necesitamos construir una máscara normalizando este kernel y ampliándolo.
- Este es un paso importante porque si no lo escala, la imagen se verá negra.
- Esto sucede porque todos los valores de píxeles estarán cerca de 0 después de superponer la máscara en la imagen de entrada.
- Después de esto, iteramos a través de todos los canales de color y aplicamos la máscara a cada canal.

```
import cv2
import numpy as np

img = cv2.imread('imagen.jpg')
rows, cols = img.shape[:2]

# Generar máscara de viñeta usando núcleos gaussianos
kernel_x = cv2.getGaussianKernel(cols,150)
kernel_y = cv2.getGaussianKernel(rows,150)
kernel = kernel_y * kernel_x.T
mask = 255 * kernel / np.linalg.norm(kernel)
output = np.copy(img)

# Aplicar la máscara a cada canal de la imagen de entrada
for i in range(3):
    output[:, :, i] = output[:, :, i] * mask

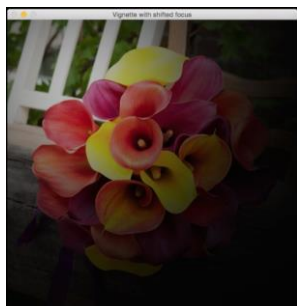
cv2.imshow('Original', img)
cv2.imshow('Vignette', output)
cv2.waitKey(0)
```

Tratamiento de imagen digital



## 7. Creación de filtro de viñeta

- Si queremos enfocarnos en una región diferente de la imagen para crear el filtro viñeta.
- Lo que tenemos que hacer es construir un núcleo gaussiano más grande y asegurarnos de que el pico coincida con la región de interés



```
import cv2
import numpy as np

img = cv2.imread('imagen.jpg')
rows, cols = img.shape[:2]

# Generar máscara de viñeta usando núcleos gaussianos
kernel_x = cv2.getGaussianKernel(int(1.5*cols),200)
kernel_y = cv2.getGaussianKernel(int(1.5*rows),200)
kernel = kernel_y * kernel_x.T
mask = 255 * kernel / np.linalg.norm(kernel)
mask = mask[int(0.5*rows):, int(0.5*cols):]
output = np.copy(img)

# Aplicar la máscara a cada canal de la imagen de entrada
for i in range(3):
    output[:, :, i] = output[:, :, i] * mask

cv2.imshow('Input', img)
cv2.imshow('Vignette with shifted focus', output)

cv2.waitKey(0)
```

Tratamiento de imagen

cv2.waitKey(0)

41



## 8. Creación de filtro de relieve

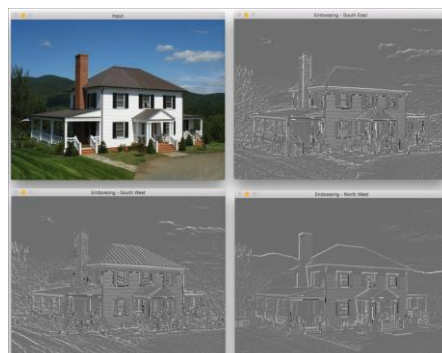
Tratamiento de imagen digital

42

## 8. Creación de filtro de relieve

Un filtro de relieve tomará una imagen y la convertirá en una imagen en relieve.

- Básicamente, tomamos cada píxel y lo reemplazamos con una sombra o un resaltado.
- Si estamos tratando con una región relativamente plana en la imagen, debemos reemplazarlo con un color gris simple porque no hay mucha información allí.
- Si hay mucho contraste en una región en particular, lo reemplazaremos con un píxel blanco (resaltado) o un píxel oscuro (sombra), dependiendo de la dirección en la que estemos grabando.



Tratamiento de imagen digital

43

## 8. Creación de filtro de relieve

Solo se reemplaza el valor de píxel actual con la diferencia de los valores de píxel vecinos en una dirección particular.

El efecto de relieve se logra compensando todos los valores de píxeles de la imagen en 128. Esta operación agrega el efecto de luces/sombras a la imagen.

```
import cv2
import numpy as np

img_emboss_input = cv2.imread('imagen.jpg')

# generating the kernels
kernel_emboss_1 = np.array([[0,-1,-1],
                             [1,0,-1],
                             [1,1,0]])
kernel_emboss_2 = np.array([[ -1,-1,0],
                             [-1,0,1],
                             [0,1,1]])
kernel_emboss_3 = np.array([[1,0,0],
                             [0,0,0],
                             [0,0,-1]])
```

```
# Convierte la imagen a escala de grises
gray_img = cv2.cvtColor(img_emboss_input,cv2.COLOR_BGR2GRAY)

# Aplica los núcleos a la imagen en escala de grises y agregar el desplazamiento
output_1 = cv2.filter2D(gray_img, -1, kernel_emboss_1) + 128
output_2 = cv2.filter2D(gray_img, -1, kernel_emboss_2) + 128
output_3 = cv2.filter2D(gray_img, -1, kernel_emboss_3) + 128

cv2.imshow('Input', img_emboss_input)
cv2.imshow('Embossing - South West', output_1)
cv2.imshow('Embossing - South East', output_2)
cv2.imshow('Embossing - North West', output_3)
cv2.waitKey(0)
```



Tratamiento de imagen digital

44