

# Practical\_2019

October 15, 2019

## 1 Procesamiento de Imágenes Digitales

Visión Computacional 2019-20 Practica 1. 3 de octubre de 2019

Autor1:Alberto Casado Garfia Autor2:Antón Cid Mejías

### 1.1 Objetivos

Los objetivos de esta práctica son:

- \* Programar algunas de las rutinas de transformaciones puntuales de procesamiento de imágenes y analizar el resultado de su aplicación.
- \* Repasar algunos conceptos de filtrado de imágenes y programar algunas rutinas para suavizado y extracción de bordes.
- \* Implementar un algoritmo de segmentación de imágenes y otro de extracción de líneas mediante la transformada de Hough.

### 1.2 Requerimientos

Para esta práctica es necesario disponer del siguiente software:

- \* Python 2.7 ó 3.X, preferiblemente el segundo
- \* Jupyter <http://jupyter.org/>.
- \* Los paquetes `python-pip` y/o `python-pip3` y el paquete "PyMaxFlow"
- \* Las librerías científicas de Python: NumPy (`python-numpy`), SciPy (`python-scipy`) y Matplotlib (`python-matplotlib`).
- \* El paquete `python-pygame`
- \* La librería OpenCV, que puedes instalar desde el paquete `python-opencv`.

Las versiones preferidas del entorno de trabajo puedes consultarlas en el Aula Virtual en el archivo "ConfiguracionPC2018.txt".

El material necesario para la práctica se puede descargar del Aula Virtual.

### 1.3 Condiciones

- La fecha límite de entrega será el martes 23 de octubre a las 23:55.
  - La entrega consiste en dos archivos con el código, resultados y respuestas a los ejercicios:
1. Un "notebook" de Jupyter con el fuente y los resultados (ejecuta "Restart & Run all" antes de guardar)
  2. Un documento "pdf" generado a partir del fuente de Jupyter, por ejemplo usando el comando `jupyter nbconvert --execute --to pdf notebook.ipynb`. Asegúrate de que el documento "pdf" contiene todos los resultados correctamente ejecutados (previamente ejecuta en el menú "Kernel" la opción "Restart & Run All").

- Las respuestas a los ejercicios debes introducirlas en tantas celdas de código o texto como creas necesarias, insertadas inmediatamente después de un enunciado y antes del siguiente.
- Las prácticas puede realizarse en parejas. Sólo es necesario que uno de los miembros del equipo entregue la práctica.

## 1.4 Instala el entorno de trabajo

1. Instala el entorno de trabajo.

```
apt install python apt install python-scipy apt install python-numpy apt
install python-matplotlib apt install python-opencv apt install jupyter apt
install jupyter-nbconvert
```

Para para trabajar con la versión 3.X de Python, basta sustituir la palabra "python" por "python3" en los comandos anteriores.

2. Instala el paquete PyMaxflow

```
pip install PyMaxflow o pip3 install PyMaxflow
```

Si no tienes el paquete "pip" debes instalarlo: apt install python-pip o apt install python3-pip 3. Instala el paquete "pygame"

```
``apt install python-pygame``
```

En Python 3.X, la versión 18.04 de Ubuntu no tiene el paquete "python3-pygame" pero puedes ins-

## 1.5 Transformaciones puntuales

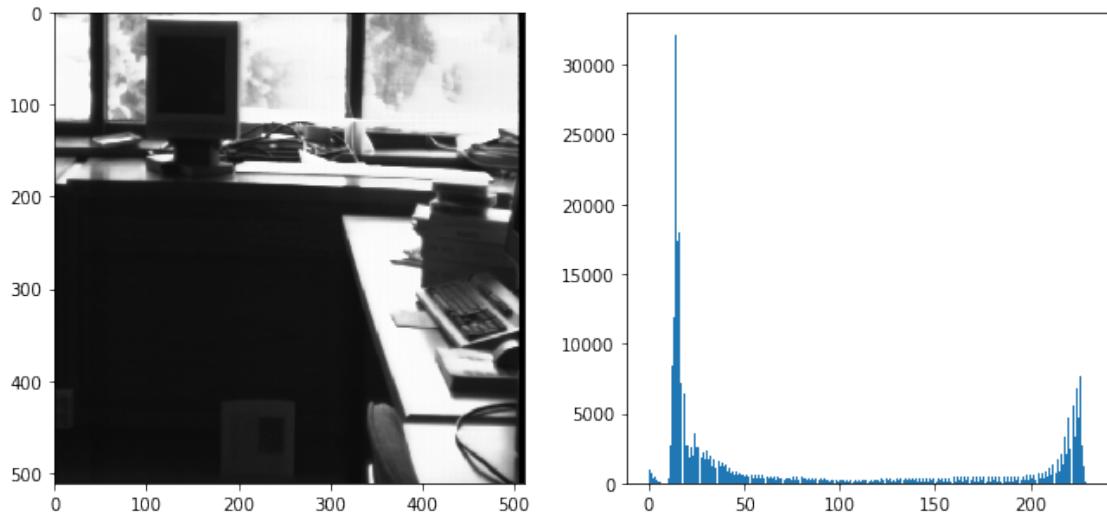
En este apartado te recomiendo que uses al menos la imagen indicada, que puedes encontrar en el directorio de imágenes del aula virtual. También puedes probar con otras que te parezcan interesantes.

**Ejercicio 1.** Carga la imagen `escilum.tif`. Calcula y muestra su histograma con la función `hist()` de `matplotlib.pyplot`. A la vista del histograma, discute qué problema tiene la imagen para analizar visualmente la región inferior izquierda.

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

```
[2]: img = cv2.imread('imagenes/escilum.tif', 0)
fig, axs = plt.subplots(1, 2, figsize=(11, 5))

axs[0].imshow(img, cmap = 'gray')
histogram = axs[1].hist(img.flatten(), bins=256)
plt.show()
```



A partir del histograma se puede ver que la mayoría de píxeles de la imagen tienen valores extremos. En el caso de la parte inferior izquierda los píxeles ocuparían el rango estimado de 0 a 50, esto es debido a que es la parte más oscura de la imagen. Puesto que este rango es muy pequeño (las diferencias entre píxeles son muy bajas), prácticamente no se aprecian los detalles del entorno.

**Ejercicio 2.** Escribe una función `eq_hist(histograma)` que calcule la función de transformación puntual que ecualiza el histograma. Aplica la función de transformación a la imagen anterior. Calcula y muestra nuevamente el histograma y la imagen resultantes. Discute los resultados obtenidos. ¿Cuál sería el resultado si volviésemos a ecualizar la imagen resultante?

```
[3]: def eq_hist(hist):

    def eq_fun(img):
        accumulative_hist = np.array([sum(hist[:i+1]) for i in
                                     range(len(hist))])
        aux = 255 / accumulative_hist[len(accumulative_hist) - 1]

        eq_img = np.vectorize(lambda p: np.uint8(round(accumulative_hist[p] * aux)))(img)

        return eq_img

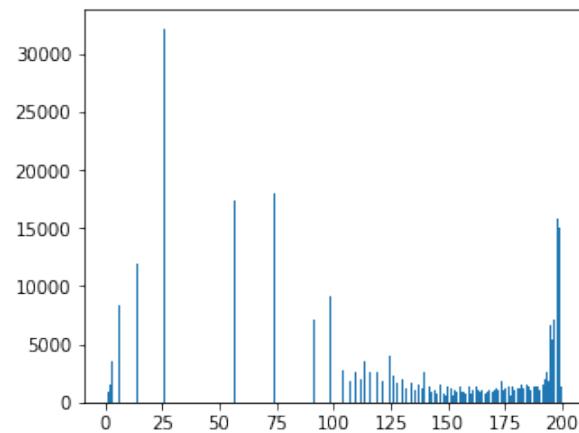
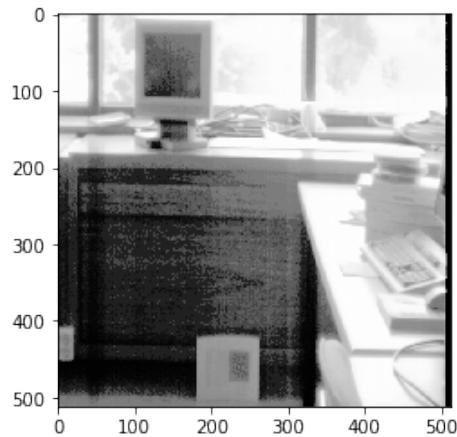
    return eq_fun

eq_fun = eq_hist(histogram[0])
eq_img = eq_fun(img)

fig, axs = plt.subplots(1, 2, figsize=(11, 4))

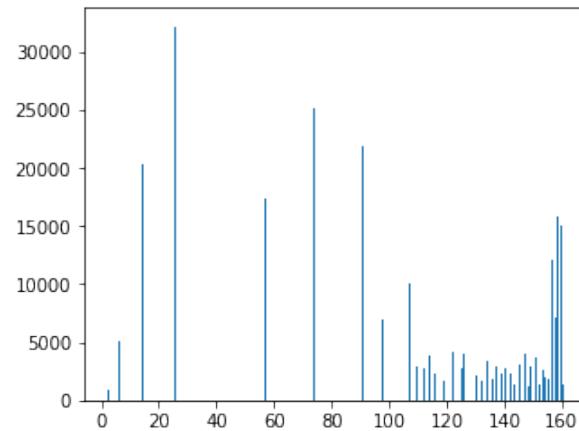
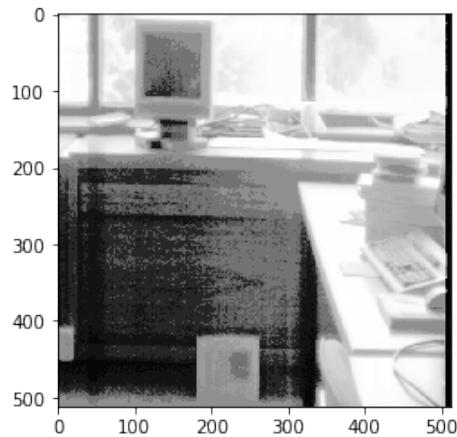
axs[0].imshow(eq_img, cmap = 'gray')
```

```
eq_histogram = axs[1].hist(eq_img.flatten(), bins=256)
plt.show()
```



```
[4]: # Vuelve a aplicar la ecualización sobre la imagen ecualizada
eq2_fun = eq_hist(eq_histogram[0])
eq2_img = eq2_fun(eq_img)

fig, axs = plt.subplots(1, 2, figsize=(11, 4))
axs[0].imshow(eq2_img, cmap = 'gray')
axs[1].hist(eq2_img.flatten(), bins=256)
plt.show()
```



En la nueva imagen obtenida se distinguen mucho mejor los detalles de la esquina inferior izquierda y de la imagen en general. Esto es debido a que al aplicar la transformación de ecualización del histograma, los valores de los píxeles que se encontraban en los extremos han adquirido valores medios. Esto provoca que se perciban mejor las diferencias de tonalidades en la imagen, y por

tanto, los detalles que antes estaban ocultos.

Si se vuelve a ecualizar un histograma ecualizado, el resultado es el mismo a el de la imagen original ecualizada. Esto es debido a que, puesto que la acumulación de histograma es la misma que en la anterior ecualización, la correspondencia entre píxeles se mantiene.

## 1.6 Filtrado

Para realizar las convoluciones utiliza la función `convolve` o `convolve1d` de `scipy.ndimage`.

Carga y muestra las imágenes `escgaus.bmp` y `escimp5.bmp` que están contaminadas respectivamente con ruido de tipo gaussiano e impulsional. En los siguientes ejercicios también puedes utilizar otras imágenes que te parezcan interesantes.

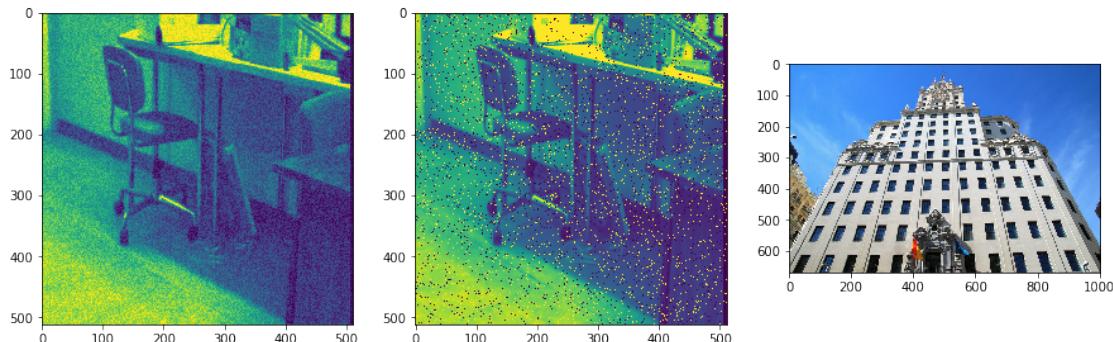
```
[5]: import numpy as np
```

```
[6]: import matplotlib.pyplot as plt
```

```
escgaus = plt.imread("imagenes/escgaus.bmp")
escimp5 = plt.imread("imagenes/escimp5.bmp")
telefonica = plt.imread("imagenes/telefonica.jpg")

fig, axs = plt.subplots(1, 3, figsize=(15, 6))

axs[0].imshow(escgaus)
axs[1].imshow(escimp5)
axs[2].imshow(telefonica)
plt.show()
```



```
[7]: from scipy.ndimage import convolve, convolve1d
```

**Ejercicio 3.** Escribe una función `masc_gaus(sigma, n)` que construya una máscara de una dimensión de un filtro gaussiano de tamaño  $n$  y varianza  $\sigma$ . Filtra las imágenes anteriores con filtros gaussianos bidimensionales de diferentes tamaños de  $n$ , y/o  $\sigma$ .

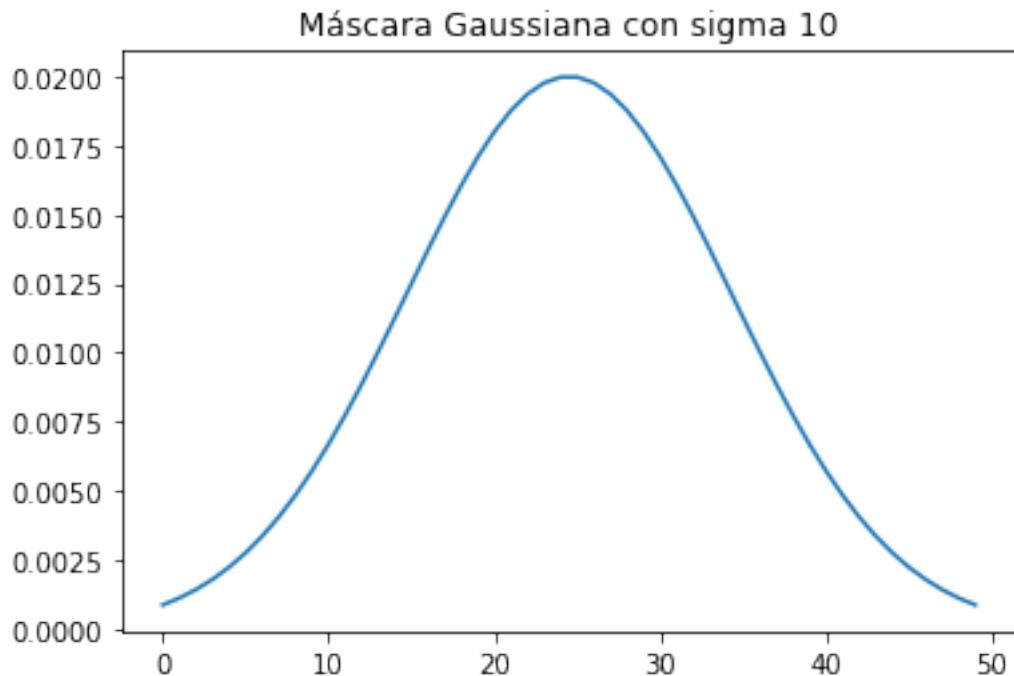
En este ejercicio tenéis que implementar vosotros la función que construye la máscara. No podéis usar funciones que construyan la máscara o realicen el filtrado automáticamente.

Muestra cómo afecta este filtrado a los dos tipos de ruido que contaminan las imágenes anteriores y discute los resultados. Pinta alguna de las máscaras utilizadas.

```
[8]: def gauss(x, sigma):
        return np.exp(-1/2 * (x/ sigma)** 2)

def masc_gaus(sigma, n):
    pos = np.linspace(-n/2, n/2, n)
    return np.array([gauss(x, sigma) / n for x in pos])

plt.title("Máscara Gaussiana con sigma 10")
plt.plot(masc_gaus(10, 50))
plt.show()
```



```
[9]: fig, axs = plt.subplots(2, 3, figsize=(15, 6))
fig.suptitle("Desenfoque Gausiano Horizontal")

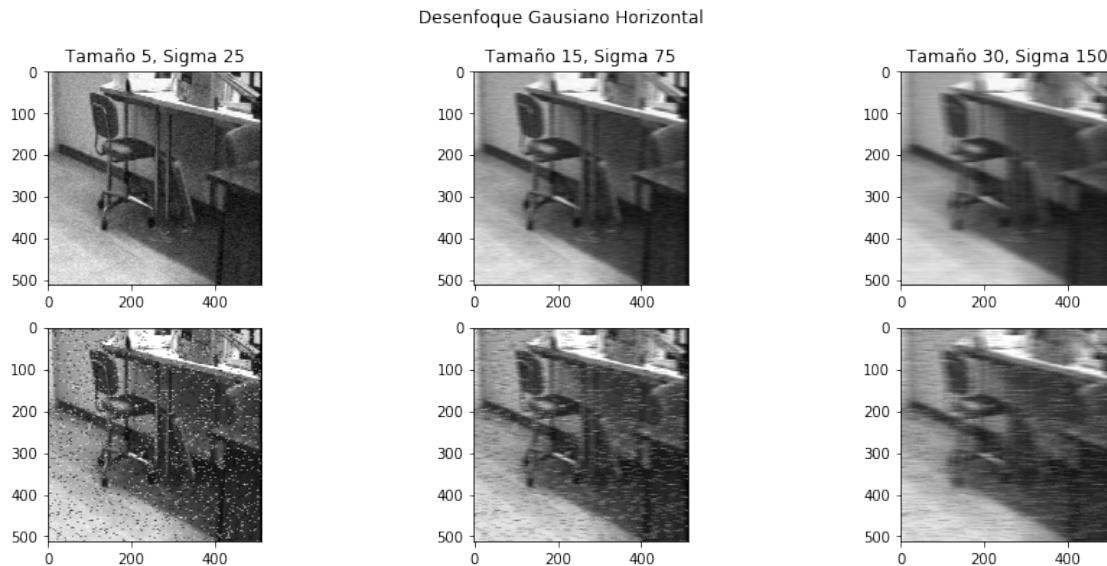
for i, x in enumerate((5, 15, 30)):
    sigma = x * 5
    mask = masc_gaus(sigma, x)
```

```

    axs[0][i].imshow(convolve1d(escgaus, weights=mask), cmap='gray', vmin=0, vmax=255)
    axs[1][i].imshow(convolve1d(escimp5, weights=mask), cmap='gray', vmin=0, vmax=255)

    axs[0][i].title.set_text('Tamaño {}, Sigma {}'.format(x, sigma))

```



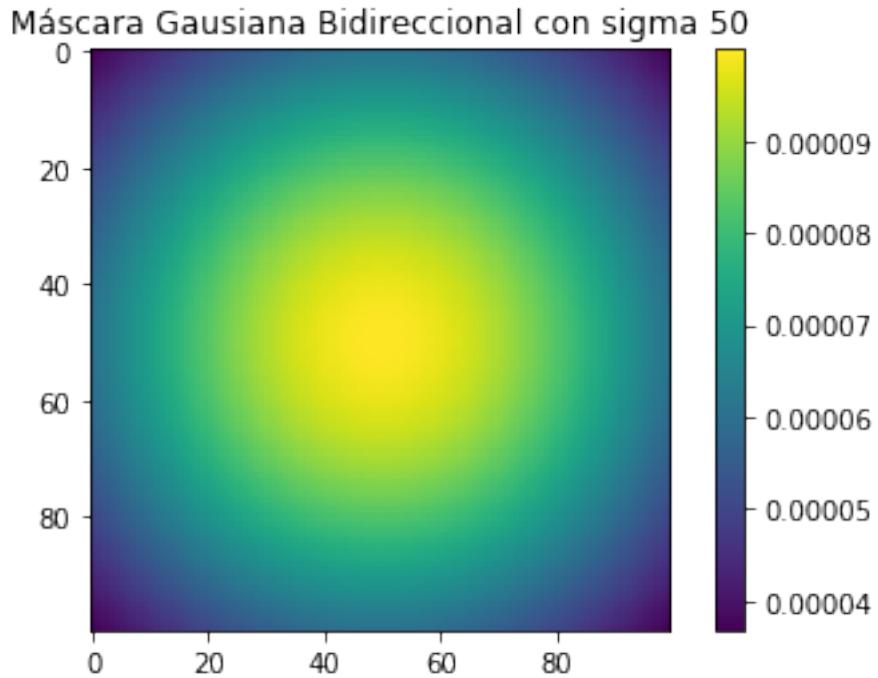
En las imágenes anteriores se puede observar como tanto la imagen como el ruido se ven desenfocados en el eje horizontal. Cuanto mayor es la sigma, mayor es el desenfoque producido.

```

[10]: def masc_gaus2D(sigma, n):
    gauss = masc_gaus(sigma, n)[np.newaxis]
    return gauss.T * gauss

ax = plt.imshow(masc_gaus2D(50, 100))
plt.title("Máscara Gausiana Bidireccional con sigma 50")
plt.colorbar(ax)
plt.show()

```



Para crear un filtro gausiano bidireccional multiplicamos dos máscaras gausianas unidireccionales de forma que nos produzcan una matriz. El resultado de esta multiplicación puede verse en la figura anterior

```
[11]: fig, axs = plt.subplots(4, 3, figsize=(15, 16))

kernels = (5, 15, 30)

axs[0][0].set_ylabel('Método A')
axs[1][0].set_ylabel('Método B')
axs[2][0].set_ylabel('Método A')
axs[3][0].set_ylabel('Método B')

for i, x in enumerate(kernels):

    sigma = x * 5

    mask2D = masc_gaus2D(sigma, x)
    mask = masc_gaus(sigma, x)

    axs[0][i].title.set_text("Tamaño {}, Sigma {}".format(x, sigma))

    axs[0][i].imshow(convolve(escgau, weights=mask2D), cmap='gray', vmin=0, vmax=255)
```

```

escgausConv = convolve1d(convolve1d(escgaus, weights=mask, axis=0),  

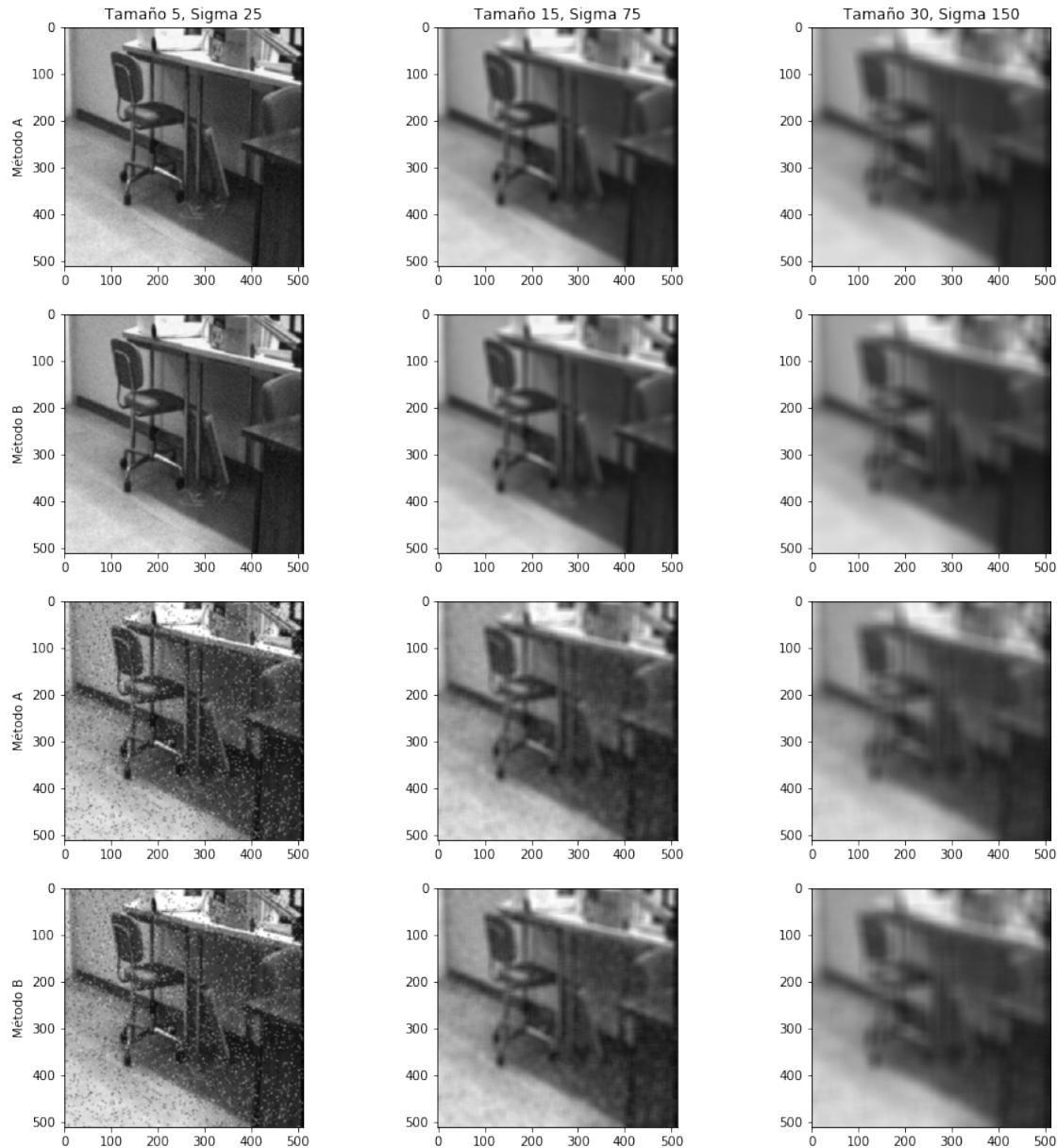
                         weights=mask, axis=1)
axs[1][i].imshow(escgausConv, cmap='gray', vmin=0, vmax=255)

axs[2][i].imshow(convolve(escimp5, weights=mask2D), cmap='gray', vmin=0,  

                  vmax=255)
escimp5Conv = convolve1d(convolve1d(escimp5, weights=mask, axis=0),  

                         weights=mask, axis=1)
axs[3][i].imshow(escimp5Conv, cmap='gray', vmin=0, vmax=255)

```



En las imágenes anteriores se puede observar el efecto del filtro gausiano bidimensional. Se ha realizado de dos formas, multiplicando las máscaras unidimensionales y luego posteriormente usando la matriz resultante en una convolución con la imagen, al que hemos llamado método A y el método B al que a la imagen se le aplican 2 convoluciones con la máscara unidimensional gausiana, una horizontal y otra vertical. Se puede observar que el resultado es el mismo, pero este último método usa menos calculos, por lo que es más eficiente.

En las imagenes de las 2 primeras filas se puede apreciar como el ruido gausiano desaparece, aunque la imagen también se ve desenfocada. En las 2 últimas filas la imagen tiene un ruido que no es gausiano, por lo que este no desaparece y solo se ve desenfocado como el resto de la imagen.

**Ejercicio 4.** Escribe una función `masc_deriv_gaus(sigma, n)` que construya una máscara de una dimensión de un filtro derivada del gaussiano de tamaño  $n$  y varianza  $\sigma^2$ . Filtra la imagen `telefonica.jpg` con filtros bidimensionales de derivada del gaussiano para extraer los bordes de la imagen. Prueba con diferentes valores de  $n$  y/o  $\sigma$ .

Muestra y discute los resultados. Pinta alguna de las máscaras construidas.

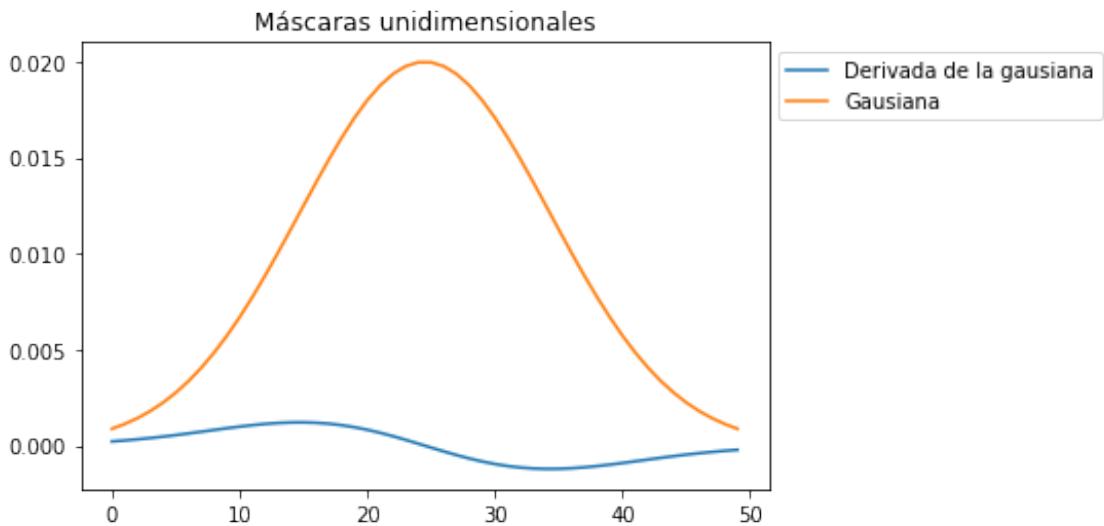
```
[12]: def deriv_gauss(x, sigma):
        return - x/sigma**2 * np.exp(-1/2 * (x/sigma)**2)

def masc_deriv_gaus(sigma, n):
    pos = np.linspace(-n/2, n/2, n)
    return np.array([deriv_gauss(x, sigma) / n for x in pos])

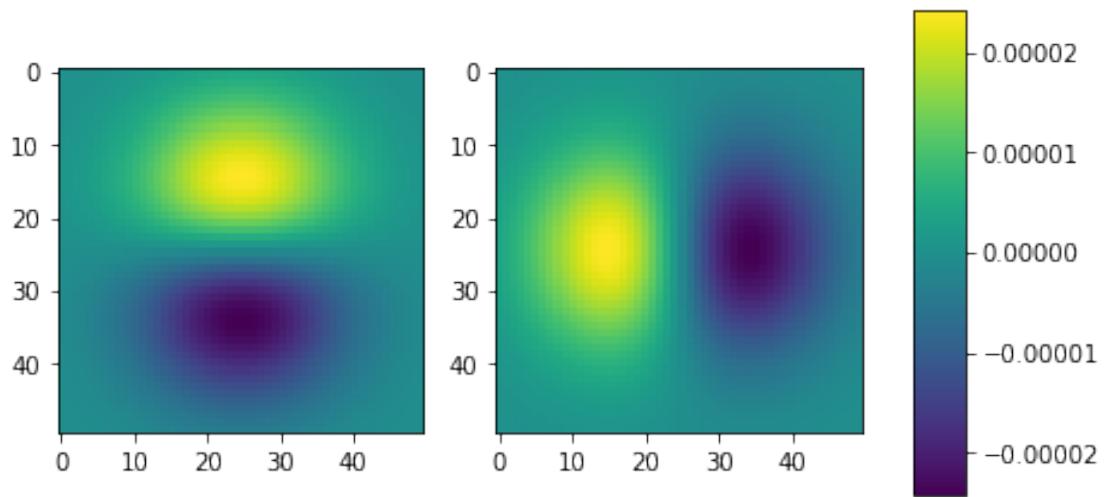
plt.title('Máscaras unidimensionales')
plt.plot(masc_deriv_gaus(10, 50), label='Derivada de la gausiana')
plt.plot(masc_gaus(10, 50), label='Gausiana')
plt.legend(loc='upper right', bbox_to_anchor=(1, 0, 0.5, 1))

img1 = masc_deriv_gaus(10, 50)[np.newaxis].T @ masc_gaus(10, 50)[np.newaxis]
img2 = masc_gaus(10, 50)[np.newaxis].T @ masc_deriv_gaus(10, 50)[np.newaxis]

fig, axs = plt.subplots(1, 2)
fig.suptitle('Máscaras de la derivada de la gausiana bidireccionales')
axs[0].imshow(img1)
im = axs[1].imshow(img2)
cbar_ax = fig.add_axes([0.95, 0.15, 0.05, 0.7])
plt.colorbar(im, cax=cbar_ax)
plt.show()
```



Máscaras de la derivada de la gausiana bidireccionales



```
[13]: def to_rgb(gray_mask):
    return np.repeat(gray_mask, 3).reshape(gray_mask.shape[0], gray_mask.
                                           shape[1], 3)
```

```
[14]: telefonica = plt.imread('./imagenes/telefonica.jpg')

def masc_deriv_gaus2DH(sigma, x):
    return masc_gaus(sigma, x)[np.newaxis].T @ masc_deriv_gaus(sigma, x)[np.
                                                               newaxis]
```

```

def masc_deriv_gaus2DV(sigma, x):
    return masc_deriv_gaus(sigma, x)[np.newaxis].T @ masc_gaus(sigma, x)[np.
    ↪newaxis]

fig, axs = plt.subplots(3, 3, figsize=(15, 10))

kernels = (3, 7, 15)
sigmas = (3, 7, 15)

i = 0

fig.suptitle("Máscaras Horizontales")

for i, x in enumerate(kernels):
    for j, sigma in enumerate(sigmas):
        mask = masc_deriv_gaus2DH(sigma, x)
        mask_rgb = to_rgb(mask)

        conv = convolve(telefonica, weights=mask_rgb)
        axs[i][j].title.set_text('Tamaño {}, Sigma {}'.format(x, sigma))
        axs[i][j].imshow(conv, vmin=0, vmax=255)

fig, axs = plt.subplots(3, 3, figsize=(15, 10))

kernels = (3, 7, 15)
sigmas = (3, 7, 15)

i = 0

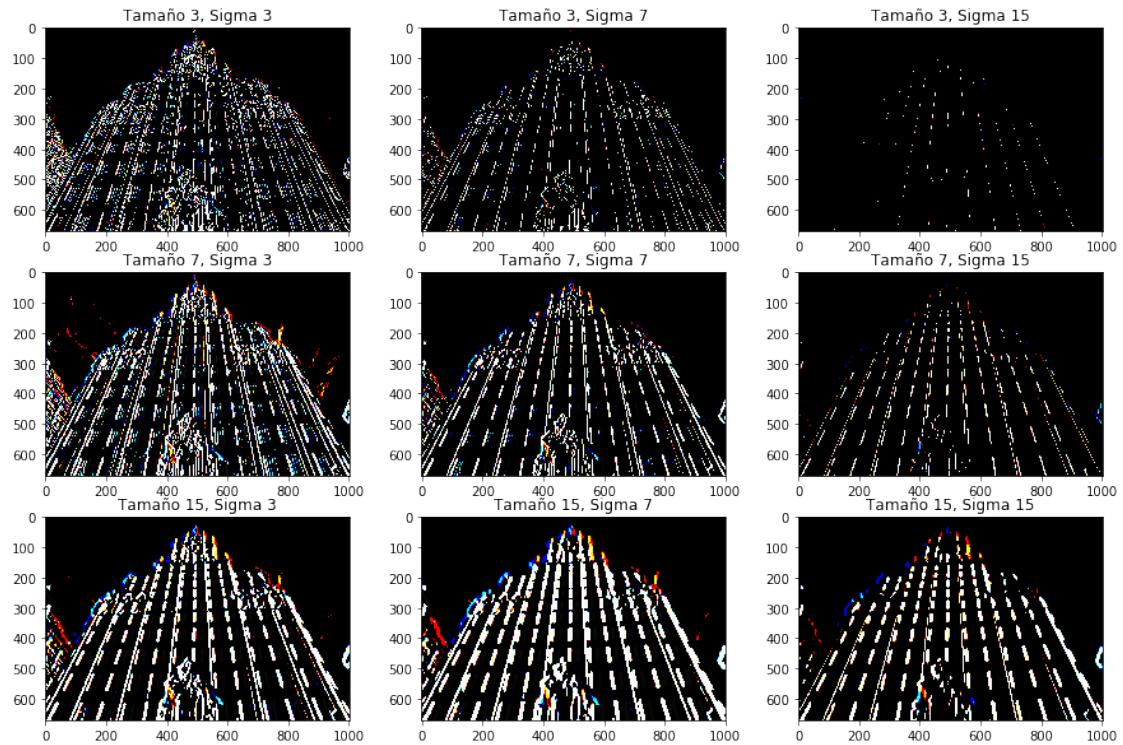
fig.suptitle("Máscaras Verticales")

for i, x in enumerate(kernels):
    for j, sigma in enumerate(sigmas):
        mask = masc_deriv_gaus2DV(sigma, x)
        mask_rgb = to_rgb(mask)

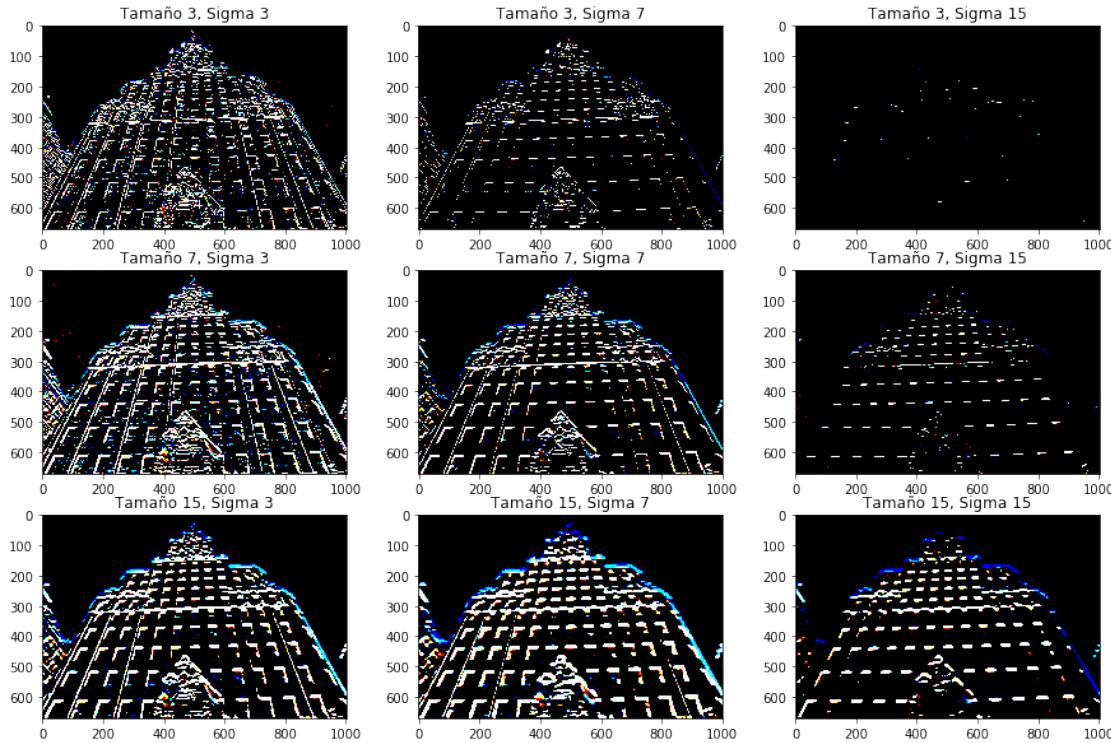
        conv = convolve(telefonica, weights=mask_rgb)
        axs[i][j].title.set_text('Tamaño {}, Sigma {}'.format(x, sigma))
        axs[i][j].imshow(conv, vmin=0, vmax=255)

```

Máscaras Horizontales



Máscaras Verticales



En las imágenes anteriores se ha aplicado el filtro horizontal y vertical bidimensional de derivada del gaussiano con diferentes sigmas y tamaños de máscara. Se puede observar que según se aumenta el tamaño de la máscara, los bordes se vuelven más gruesos y los bordes menos importantes comienzan a desaparecer. Cuando el parámetro sigma crece el filtro empieza a solo dibujar los bordes más importantes. En estos casos nos referimos a bordes importantes a aquellos que están formados por grandes diferencias entre el color de los píxeles.

```
[15]: def masc_deriv_bidireccional(img, sigma, x):
    maskh = to_rgb(masc_deriv_gaus2DH(sigma, x))
    maskv = to_rgb(masc_deriv_gaus2DV(sigma, x))

    conv_h1 = convolve(img, weights= maskh).astype('uint16')
    conv_v1 = convolve(img, weights= maskv).astype('uint16')
    conv_h2 = convolve(img, weights=-maskh).astype('uint16')
    conv_v2 = convolve(img, weights=-maskv).astype('uint16')

    # plt.imshow(conv_h1)
    # plt.show()
    # plt.imshow(conv_v1)
    # plt.show()
    # plt.imshow(conv_h2)
```

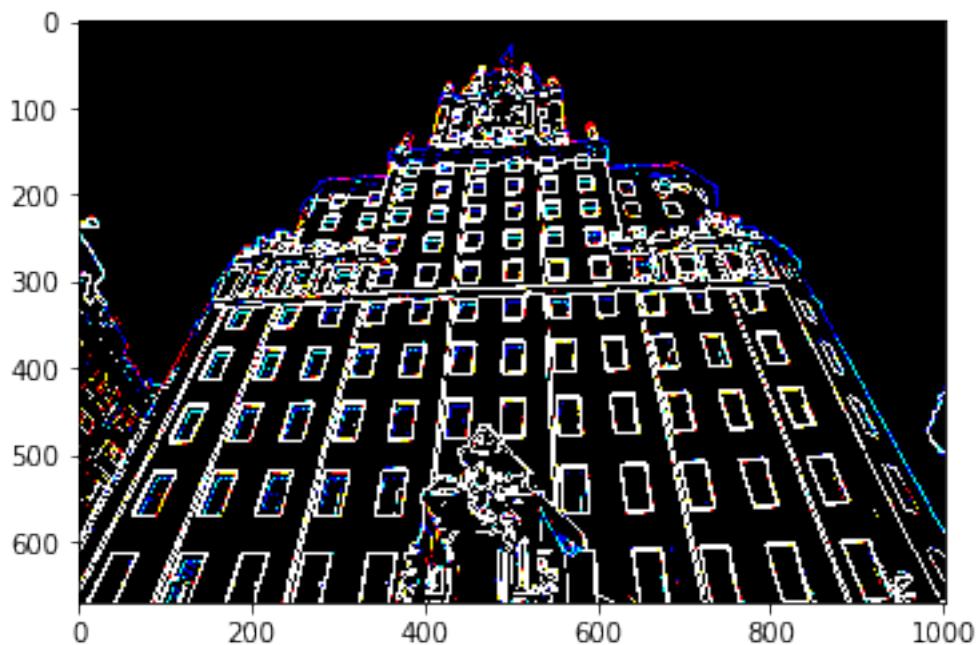
```

#      plt.show()
#      plt.imshow(conv_v2)
#      plt.show()

    return np.clip(conv_h1 + conv_h2 + conv_v1 + conv_v2, 0, 255).
→astype('uint8')

img = masc_deriv_bidireccional(telefonica, 10, 5)
plt.imshow(img)
plt.show()

```



Usando las máscaras horizontales y verticales, calculando los opuestos de las máscaras, convolucionando la imagen con las máscaras resultantes y, finalmente, sumándolas obtenemos una imagen donde se resaltan todos los bordes.

**Ejercicio 5.** Utiliza la función `median_filter` del paquete `scipy.ndimage` que realice el filtrado de la imagen con un filtro de la mediana de tamaño  $n \times n$ .

Muestra y discute los resultados para diferentes valores del parámetro  $n$  en ambas imágenes. Comáralos con los obtenidos en el Ejercicio 3.

```
[16]: from scipy.ndimage import median_filter

escgaus = plt.imread("imagenes/escgaus.bmp")
escimp5 = plt.imread("imagenes/escimp5.bmp")
```

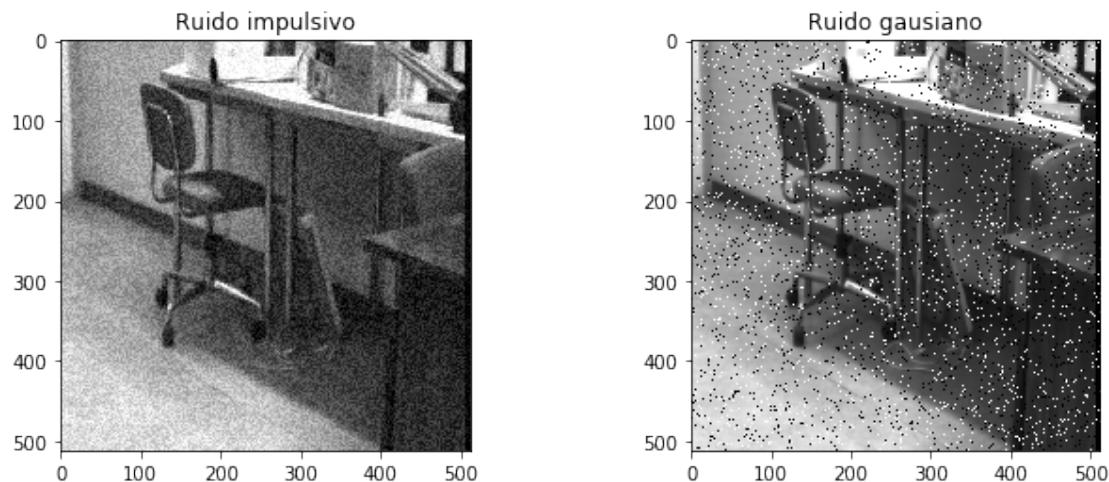
```

fig, axs = plt.subplots(1, 2, figsize=(11, 4))

axs[0].title.set_text('Ruido impulsivo')
axs[1].title.set_text('Ruido gausiano')

axs[0].imshow(escgaus, cmap = 'gray')
axs[1].imshow(escimp5, cmap = 'gray')
plt.show()

```



```

[17]: fig, axs = plt.subplots(5, 2, figsize=(12, 25))
n = [1, 3, 5, 9, 13]

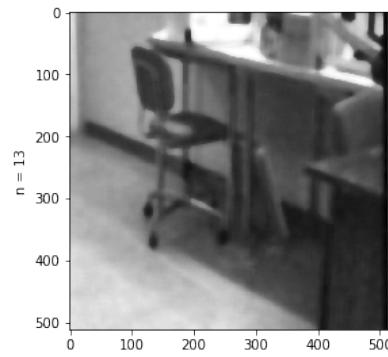
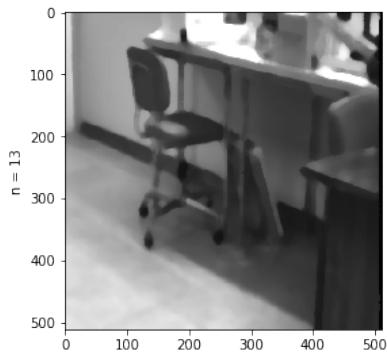
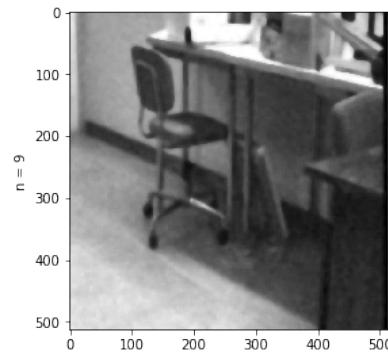
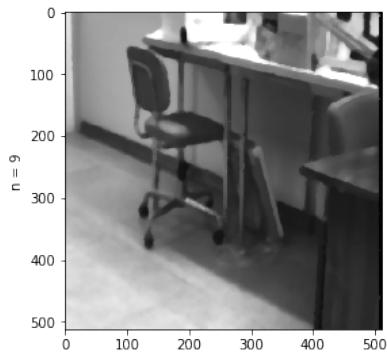
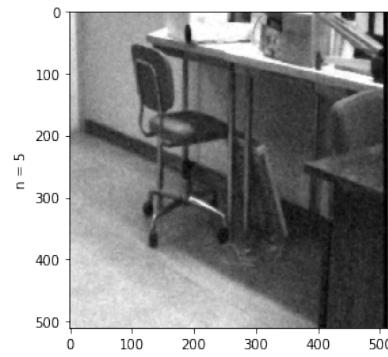
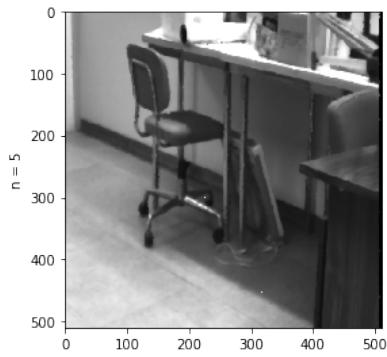
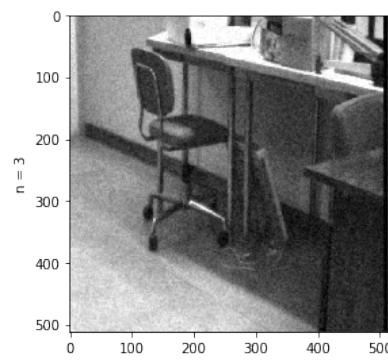
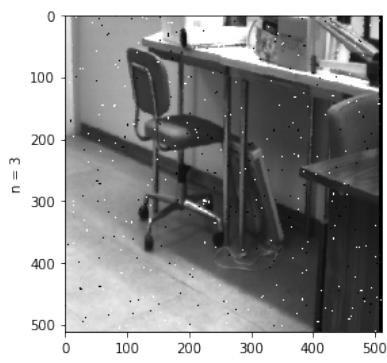
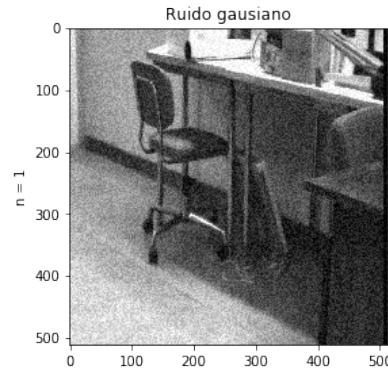
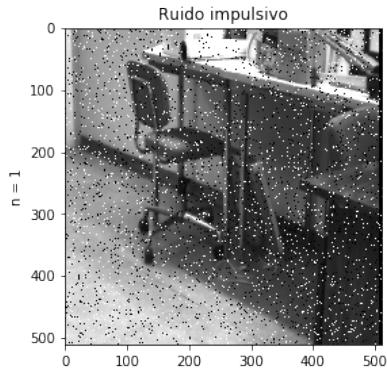
axs[0][0].title.set_text('Ruido impulsivo')
axs[0][1].title.set_text('Ruido gausiano')

for i in range(len(n)):

    axs[i][0].set_ylabel('n = %i' % n[i])
    axs[i][1].set_ylabel('n = %i' % n[i])
    axs[i][0].imshow(median_filter(escimp5, size = n[i]), cmap='gray')
    axs[i][1].imshow(median_filter(escgaus, size = n[i]), cmap='gray')

plt.show()

```



Se ha aplicado un filtro de mediana con diferentes tamaños de ventana.

En el caso del ruido impulsivo, con un tamaño de  $n = 3$  se elimina la mayoría de ruido sin difuminar prácticamente la imagen. Si subimos ese valor a 5 conseguimos eliminar la totalidad del ruido, a excepción de algunos puntos restantes. A partir de este último valor se elimina completamente el ruido a costa de la difuminación de la imagen.

El ruido gausiano no se logra eliminar para los tamaños de 3 y 5. A partir de estos tamaños, aunque aún algo presente, empieza a desaparecer. En comparación al ruido impulsivo, la imagen se difumina más al aumentar el tamaño de  $n$ . Este tipo de filtro elimina muy bien el ruido impulsivo puesto que, al calcular la mediana de una ventana de píxeles, los valores irregulares se eliminan.

En relación al filtro gaussiano, se puede visualizar, a partir de las imágenes, la manera distinta de actuar que tiene cada uno. Mientras que el filtro gaussiano esparce el ruido impulsivo a los vecinos, el de mediana lo elimina. Pasa al contrario con el ruido gausiano, donde es el filtro de mediana el que incrementa el ruido.

**Ejercicio 6.** Utiliza la función `cv2.bilateralFilter()` de OpenCV para realizar el filtrado bilateral de una imagen. Selecciona los parámetros adecuados y aplícalo a las imágenes `tapiz.png`, `escgaus.bmp` y `escimp5.bmp` y otras que elijas tú.

Si llamamos  $\sigma_r$  a la varianza de la gaussiana que controla la ponderación debida a la diferencia entre los valores de los píxeles y  $\sigma_s$  a la varianza de la gaussiana que controla la ponderación debida a la posición de los píxeles. Responde a las siguientes preguntas:

- \* ¿Cómo se comporta el filtro bilateral cuando la varianza  $\sigma_r$  es muy alta? ¿En este caso qué ocurre si  $\sigma_s$  es alta o baja?
- \* ¿Cómo se comporta si  $\sigma_r$  es muy baja? ¿En este caso cómo se comporta el filtro dependiendo si  $\sigma_s$  es alta o baja?

Muestra y discute los resultados para distintos valores de los parámetros, tanto para las imágenes contaminadas con ruido gausiano como impulsivo. Compáralos con los obtenidos en los Ejercicios 3 y 5.

```
[18]: import cv2

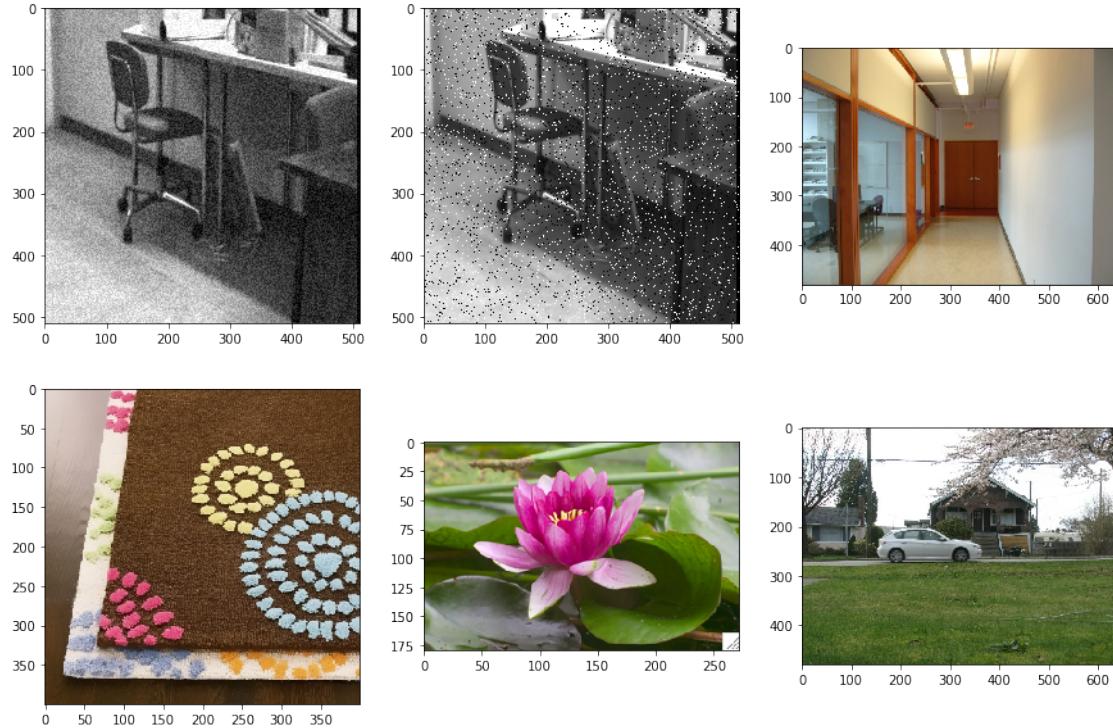
escgaus = plt.imread("imagenes/escgaus.bmp")
escimp5 = plt.imread("imagenes/escimp5.bmp")
tapiz = plt.imread("imagenes/tapiz.png")
flower = plt.imread("imagenes/flower.png")
corridor = plt.imread("imagenes/corridor.jpg")
car = plt.imread("imagenes/car.jpg")

fig, axs = plt.subplots(2, 3, figsize=(15, 10))

axs[0][0].imshow(escgaus, cmap='gray')
axs[0][1].imshow(escimp5, cmap='gray')
axs[0][2].imshow(corridor)
axs[1][0].imshow(tapiz)
axs[1][1].imshow(flower)
```

```
axs[1][2].imshow(car)

plt.show()
```



```
[19]: fig, axs = plt.subplots(6, 2, figsize=(16, 32))

axs[0][0].title.set_text('Original')
axs[0][1].title.set_text('Filtro bilateral')

axs[0][0].imshow(escgaus, cmap='gray')
axs[0][1].imshow(cv2.bilateralFilter(escgaus, 7, 70, 0), cmap='gray')

axs[1][0].imshow(escimp5, cmap='gray')
axs[1][1].imshow(cv2.bilateralFilter(escimp5, 7, 80, 10), cmap='gray')

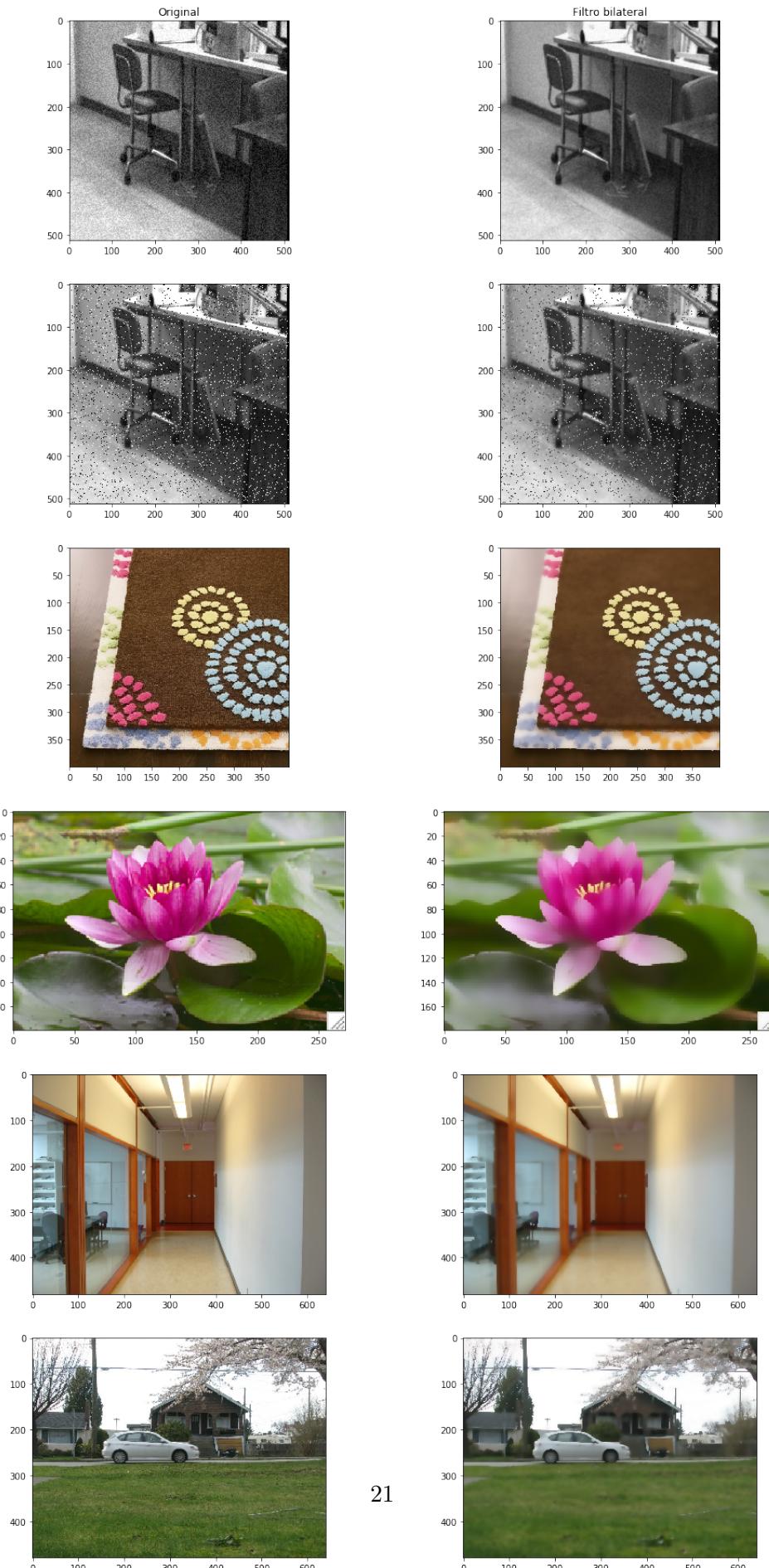
axs[2][0].imshow(tapiz)
axs[2][1].imshow(cv2.bilateralFilter(tapiz, -1, 0.5, 4))

axs[3][0].imshow(flower)
axs[3][1].imshow(cv2.bilateralFilter(flower, -1, 0.5, 4))

axs[4][0].imshow(corr)
axs[4][1].imshow(cv2.bilateralFilter(corr, 11, 120, 200))
```

```
axs[5][0].imshow(car)
axs[5][1].imshow(cv2.bilateralFilter(car, 11, 120, 200))

plt.show()
```



```
[20]: fig, axs = plt.subplots(4, 2, figsize=(15, 22))

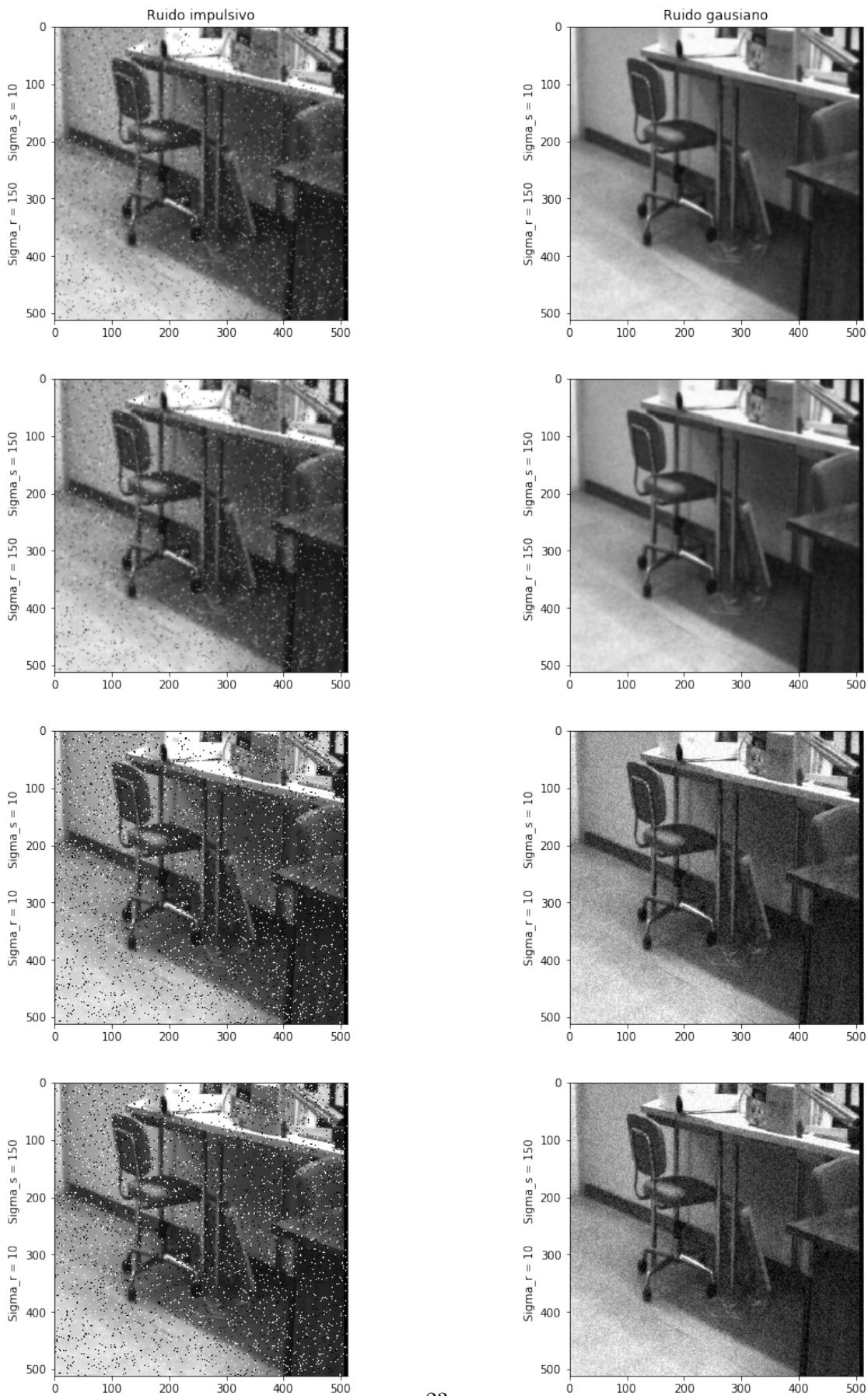
axs[0][0].title.set_text('Ruido impulsivo')
axs[0][1].title.set_text('Ruido gausiano')

sigma_r = [150, 150, 10, 10]
sigma_s = [10, 150, 10, 150]

for i in range(4):

    axs[i][0].set_ylabel('Sigma_r = %i' % (sigma_r[i], sigma_s[i]))
    axs[i][1].set_ylabel('Sigma_r = %i' % (sigma_r[i], sigma_s[i]))
    axs[i][0].imshow(cv2.bilateralFilter(escimp5, 7, sigma_r[i], sigma_s[i]), cmap='gray')
    axs[i][1].imshow(cv2.bilateralFilter(escgaus, 7, sigma_r[i], sigma_s[i]), cmap='gray')

plt.show()
```



Se han ajustado los valores, tanto de Sigma\_s como de Sigma\_r, para extraer regiones y eliminar ruido. El resultado ha sido satisfactorio en todos los casos excepto con el ruido impulsivo.

A continuación, se han aplicado filtros bilaterales sobre las imágenes con ruido impulsivo y gaussiano. En esta prueba se ha aplicado el filtro estableciendo valores extremos tanto de Sigma\_s como de Sigma\_r:

- Si Sigma\_r alto y Sigma\_s bajo, el filtro actúa como un filtro gaussiano convencional, se elimina el ruido gaussiano y se difumina el impulsivo.
- Si Sigma\_r alto y Sigma\_s alto, se suavizan los píxeles con vecinos parecidos y se preservan los demás. Los preservados van a ser los pertenecientes a bordes o ruido impulsivo.
- Si Sigma\_r bajo y Sigma\_s bajo, no se modifica prácticamente la imagen. Se suavizarán de manera muy débil las texturas.
- Si Sigma\_r bajo y Sigma\_s alto, no se modificará prácticamente la imagen. Los píxeles de colores similares se influenciarán en gran cantidad pero el suavizado será muy bajo.

## 1.7 Transformada Hough

**Ejercicio 7.** Utiliza la función cv2.HoughLines() de OpenCV para encontrar líneas en la imagen telefonica.jpg. Para extraer los bordes de la imagen utiliza las funciones escritas más arriba.

Discute el funcionamiento para distintos valores de los parámetros de la función, así como de los filtros utilizados para extraer los bordes de la imagen. Pinta los resultados sobre la imagen (te proporcionamos algo de código por si fuese útil).

```
[21]: import cv2
import math
import matplotlib.pyplot as plt
import numpy as np

telefonica = plt.imread("imagenes/telefonica.jpg")

def draw_lines(img, lines, color=(0, 0, 255), thickness=2):
    """
    Draws a set of lines detected using the OpenCV Hough transform
    :param img: An input image in BGR format of type np.int8
    :param lines: List or Numpy array containing the parameters of the
    ↪homogeneous line as: ax + by + c = 0
    :param color: The color used to draw the lines. Red by default.
    :param thickness: The thickness of the lines to be drawn
    """
    if lines is not None:
        for i in range(len(lines)):
            eq = lines[i]
            rho = -eq[2]
            a = eq[0]
```

```

    b = eq[1]
    x0 = a * rho
    y0 = b * rho
    x1 = int(x0 - 1000 * b)
    y1 = int(y0 + 1000 * a)
    x2 = int(x0 + 1000 * b)
    y2 = int(y0 - 1000 * a)

    cv2.line(img, (x1, y1), (x2, y2), color, thickness)

#####
sigma = [10, 10, 10, 10, 10, 5, 12, 12]
n = [5, 5, 5, 5, 3, 7, 7]
threshold = [180, 180, 180, 180, 180, 180, 180, 180]
rho = [1, 1, 1, 3, 1, 1, 1, 1]
theta = [np.pi/180, np.pi/360, np.pi/180, np.pi/180, np.pi/180, np.pi/180, np.pi/180, np.pi/180, np.pi/90]
line_threshold = [350, 350, 400, 700, 350, 350, 340, 340]

fig, axs = plt.subplots(8, 2, figsize=(15, 30))

for i in range(8):
    '''
    sigma = sigmas[i]
    n = ns[i]
    # Máscara derivada gausiana horizontal
    mask_h = masc_deriv_gaus2DH(sigma, n)
    mask_h_rgb = to_rgb(mask_h)
    # Máscara derivada guasiana vertical
    mask_v = masc_deriv_gaus(sigma, n)[np.newaxis].T @ masc_gaus(sigma, n)[np.newaxis]
    mask_v_rgb = to_rgb(mask_v)'''

    # Convoluciona con ambas máscaras para obtener los bordes verticales
    # y horizontales
    #edges_v = convolve(telefonica, weights=mask_v_rgb)
    #edges_h = convolve(telefonica, weights=mask_h_rgb)
    #edges = edges_v + edges_h
    edges = masc_deriv_bidireccional(telefonica, sigma[i], n[i])

    edges_gray = cv2.cvtColor(edges, cv2.COLOR_RGB2GRAY)
    edges_threshold = np.array(edges_gray > threshold[i]).astype('uint8') * 255

    lines = np.squeeze(cv2.HoughLines(edges_threshold, rho[i], theta[i], line_threshold[i]))
    # Convert the lines to homogeneous coordinates

```

```

lines = np.array([np.cos(lines[:, 1]), np.sin(lines[:, 1]), -lines[:, 0]]).T

# Draw and show the lines
telefonica_copy = telefonica.copy()
draw_lines(telefonica_copy, lines)

axs[i][0].set_xlabel('Sigma = %.2f, n = %i, threshold = %i' % (sigma[i],  

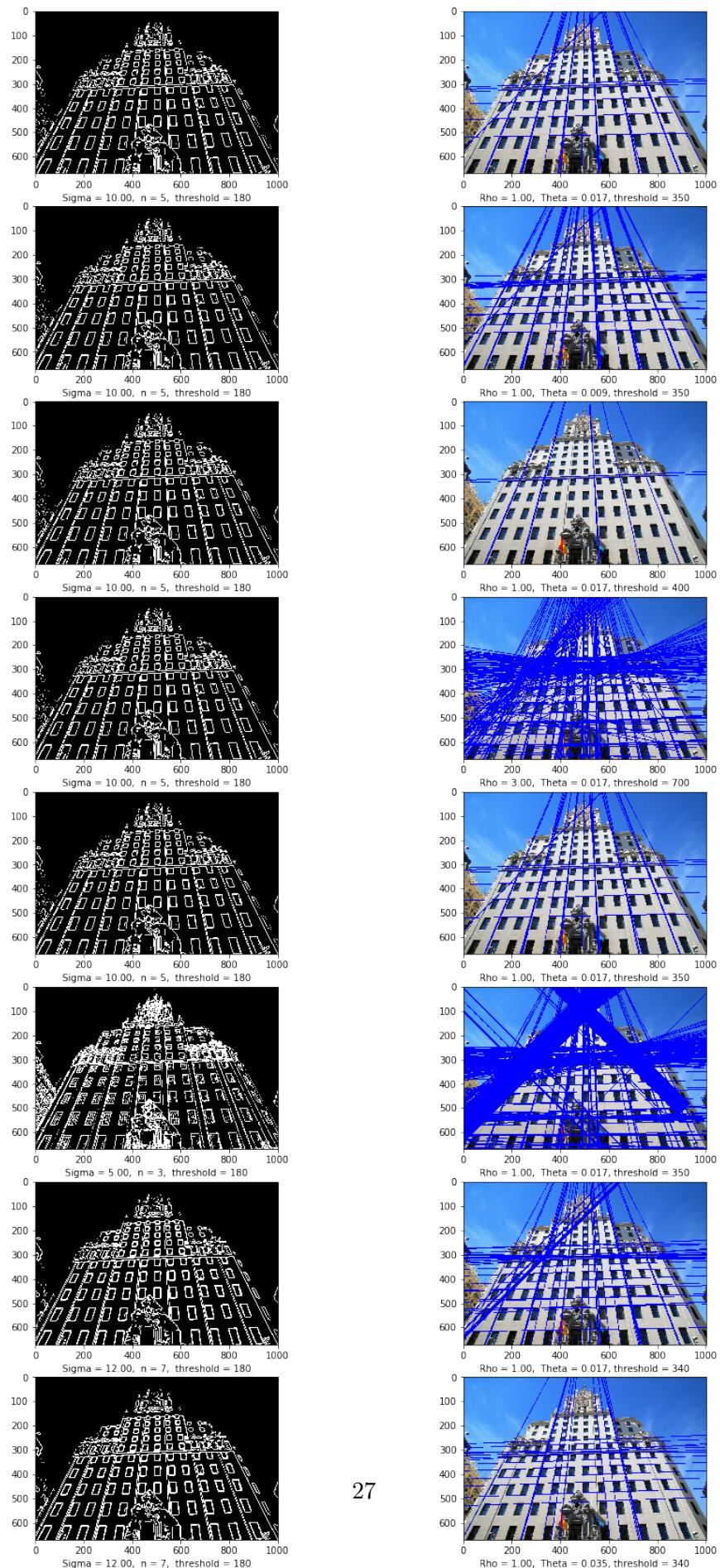
→n[i], threshold[i]))
axs[i][0].imshow(edges_threshold, cmap='gray')

axs[i][1].set_xlabel('Rho = %.2f, Theta = %.3f, threshold = %i' % (rho[i],  

→theta[i], line_threshold[i]))
axs[i][1].imshow(telefonica_copy)

plt.show()

```



Se ha hecho uso de las derivadas del gaussiano junto con un threshold, para extraer los bordes de la imagen. En las 5 primeras imágenes se ha usado un sigma de valor 10 y ventana 5. Posteriormente se han obtenido los bordes con un sigma menor y otro mayor al original.

En relación a la transformada de Hough se han probado varios valores. El el último intento se puede ver como utilizando un threshold de 340, un theta de 0.035 ( $\pi/90$ ) y un valor de rho de 1, se ha obtenido un resultado bastante bueno. Al aumentar el valor de theta, se disminuye los ángulos posibles de detección de líneas. Al aumentar rho, de detecta un mayor número de líneas. Con el threshold se indica a partir de que valor de votos de detecta una línea.

## 1.8 Segmentación

**Ejercicio 10.** Escribe una función que segmente el objeto central de una imagen a partir de una segmentación manual inicial realizada por el usuario. Puedes utilizar el código proporcionado en el archivo `segm.py`. En la optimización 1. toma como afinidad entre una pareja de píxeles la diferencia en sus valores de color y; 2. sólo establece los términos unitarios de los píxeles marcados por el usuario.

Apícalo al menos a las imágenes `persona.png` y `horse.jpg`. Muestra y discute los resultados.

```
[22]: #####
# Segmentacion de imagen a la "Grab Cut" simplificado
# por Luis Baumela. UPM. 15-10-2015
# Vision por Computador. Master en Inteligencia Artificial
#####

import numpy as np
import maxflow
import matplotlib.pyplot as plt
import sys
sys.path.insert(1, './code')
import select_pixels as sel

def segment(imgName, func):
    img = plt.imread(imgName)

    print(img.shape)

    # Marco algunos píxeles que pertenecen el objeto y el fondo
    markedImg = sel.select_fg_bg(img)

    # Create the graph.
    g = maxflow.Graph[float]()
    
```

```

# Add the nodes. nodeids has the identifiers of the nodes in the grid.
nodeids = g.add_grid_nodes(img.shape[:2])

h, w = img.shape[:2]

# Calcula los costes de los nodos no terminales del grafo
# Estos son los costes de los vecinos horizontales
exp_aff_h = np.ones(img.shape[:2])
# Estos son los costes de los vecinos verticales
exp_aff_v = np.ones(img.shape[:2])
for i in range(0, h):
    for j in range(1, w):
        exp_aff_h[i, j] = func(img[i][j], img[i][j-1])
        exp_aff_v[i, j] = func(img[i][j], img[i-1][j])

# Construyo el grafo
# Para construir el grafo relleno las estructuras
hor_struc=np.array([[0, 0, 0],
                    [1, 0, 0],
                    [0, 0, 0]])

ver_struc=np.array([[0, 1, 0],
                    [0, 0, 0],
                    [0, 0, 0]])

# Construyo el grafo
g.add_grid_edges(nodeids, exp_aff_h, hor_struc,symmetric=True)
g.add_grid_edges(nodeids, exp_aff_v, ver_struc,symmetric=True)

# Leo los pixeles etiquetados
# Los marcados en rojo representan el objeto
pts_fg = np.transpose(np.where(np.all(np.equal(markedImg,(255,0,0)),2)))
# Los marcados en verde representan el fondo
pts_bg = np.transpose(np.where(np.all(np.equal(markedImg,(0,255,0)),2)))

# Incluyo las conexiones a los nodos terminales
# Pesos de los nodos terminales
weight_default = -np.inf
g.add_grid_tedges(nodeids[pts_fg[:,0],pts_fg[:,1]], weight_default, np.inf)
g.add_grid_tedges(nodeids[pts_bg[:,0],pts_bg[:,1]], np.inf, weight_default)

# Find the maximum flow.
g.maxflow()
# Get the segments of the nodes in the grid.
sgm = g.get_grid_segments(nodeids)

# Muestro el resultado de la segmentacion

```

```

plt.figure()
plt.imshow(np.uint8(np.logical_not(sgm)),cmap='gray')
plt.show()

# Lo muestro junto con la imagen para ver el resultado
plt.figure()
wgs=(np.float_(np.logical_not(sgm))+0.3)/1.3

# Replico los pesos para cada canal y ordeno los indices
wgs=np.rollaxis(np.tile(wgs,(3,1,1)),0,3)
plt.imshow(np.uint8(np.multiply(img,wgs)))
plt.show()

```

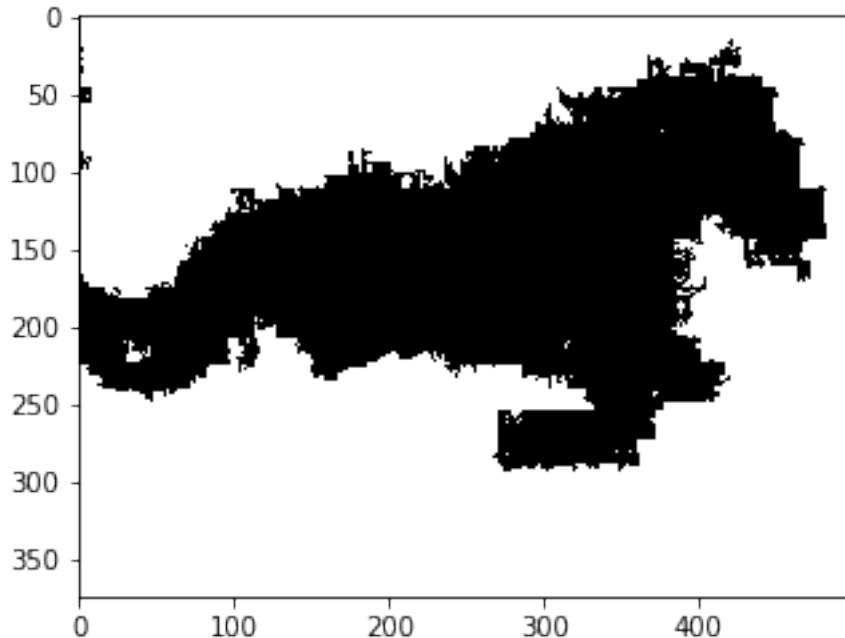
pygame 1.9.6

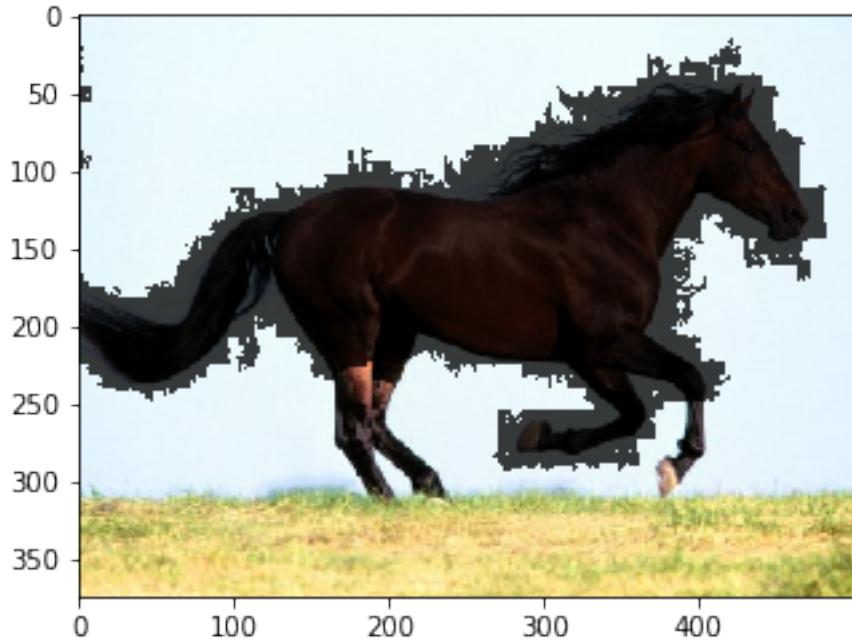
Hello from the pygame community. <https://www.pygame.org/contribute.html>

```
[26]: def default(ip, iq):
    factor = .7
    return np.abs(ip - iq).mean() * factor

segment('imagenes/horse.jpg', default)
```

(375, 500, 3)





Como se puede observar el caballo ha sido segmentado, pero delimitandolo de una manera muy imprecisa. Esto es debido a que hemos usado como función de afinidad la diferencia de los pixeles. La media se aplica para convertir los valores rgb a un solo valor en escala de grises.

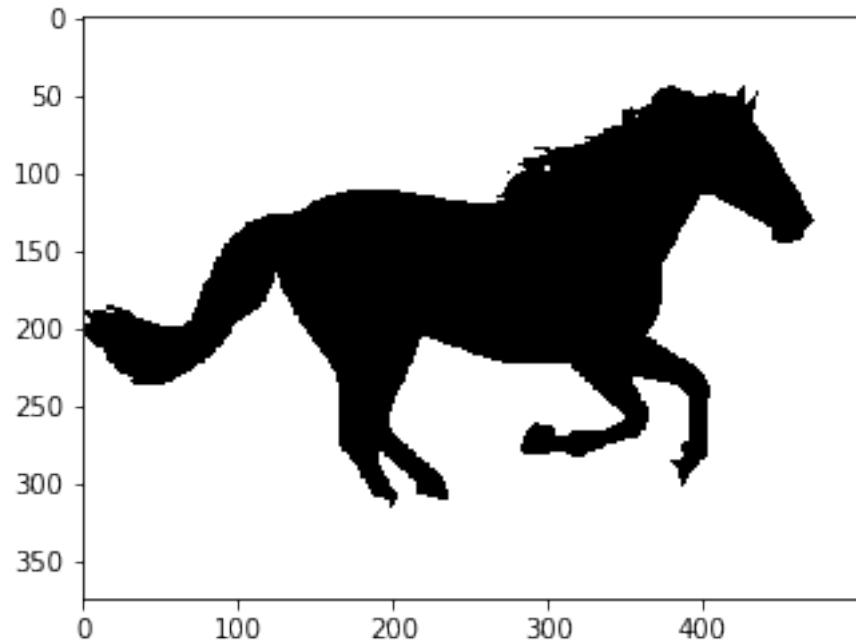
**Ejercicio 11.** Mejora el algoritmo anterior. Sugerencia: \* Refina la segmentación iterativamente.  
\* Mejora la función de afinidad entre píxeles. \* Mejora los términos unitarios

mejora los resultados de algunas de las imágenes anteriores. Muestra y discute los resultados.

```
[24]: def bpq(ip, iq):
    sigma = 0.6
    return np.exp(- (ip.mean() - iq.mean())**2 / 2*sigma**2)

segment('imagenes/horse.jpg', bpq)
# segment('imagenes/persona.png', bpq) # Falla
```

(375, 500, 3)



En este ejemplo hemos usado la función Boundary Costs descrita en el paper GraphCuts. No se incluye el segundo termino de la multiplicación porque la distancia entre píxeles es 1 y se realiza la media para obtener el nivel de gris.

```
[25]: ranges = np.linspace(-1, 1, 100)

plt.plot(ranges, [bpq(np.array([x, x, x]), np.zeros(3)) for x in ranges],  
         label='Pbq')
plt.plot(ranges, [default(np.array([x, x, x]), np.zeros(3)) for x in ranges],  
         label='Default')
plt.legend()
plt.show()
```

