Lab 3

- Filters: filtering and noise attenuation / removal.
- The Sobel operator: computing the image gradient.
- The Canny detector: contour segmentation.
- Region segmentation using Flood-Filling.

3.1 Averaging Filters

Compile and test the file **Aula_03_exe_01.py**

Analyze the code and verify how an averaging filter is applied using the function:

```
cv2.blur(src, ksize[, dst[, anchor[, borderType]]]) → dst
```

Write additional code allowing to:

- Apply (5 × 5) and (7 × 7) averaging filters to a given image.
- Apply successively (e.g., 3 times) the same filter to the resulting image.
- Visualize the result of the successive operations.

Test the developed operations using the **Lena_Ruido.png** and **DETI_Ruido.png** images.

TASK

Analyze the effects of applying different **averaging filters** to various images, and to compare the resulting images among themselves and with the original image.

Use the following test images:

- **fce5noi3.bmp**
- **fce5noi4.bmp**
- **fce5noi6.bmp**
- **sta2.bmp**
- **sta2noi1.bmp**

3.2 Median Filters

Create a new example (**Aula_03_exe_02.py**) that allows, similarly to the previous example, applying median filters to a given image.

Use the function:

cv2.medianBlur(src, ksize[, dst]) → dst¶

Test the developed operations using the **Lena_Ruido.png** and **DETI_Ruido.png** images.

TASK

Use the developed code to analyze the effects of applying different **median filters** to various images, and to compare the resulting images among themselves and with the original image, as well as with the results of applying **averaging filters**.

Use the same test images as before.

3.3 Gaussian Filters

Create a new example (**Aula_03_exe_03.py**) that allows, similarly to the previous example, applying Gaussian filters to a given image.

Use the function:

cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst

Test the developed operations using the **Lena_Ruido.png** and **DETI_Ruido.png** images.

TASK

Use the developed code to analyze the effects of applying different **Gaussian filters** to various images, and to compare the resulting images among themselves and with the original image, as well as with the results of applying **averaging filters** and **median filters**.

Use the same test images as before.

3.4 Computing the image gradient using the Sobel Operator

Compile and test the file **Aula_03_exe_04.py**

Analyze the code and verify how the Sobel operator is applied, to compute the first order directional derivatives, using the function:

```
cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]]) → dst
```

Note the following:

- The resulting image uses a signed, 64-bit representation for each pixel.
- A conversion to the usual gray-level representation (8 bits, unsigned) is required for a proper display.

TASK

Write additional code to allow applying the (5 × 5) Sobel operator.

And to combine the two directional derivatives using:

$$result = GradientX^2 + GradientY^2$$

where *GradientX* and *GradientY* represent the directional derivatives computed with the Sobel operator.

Test the developed operations using the **wdg2.bmp**, **lena.jpg, cln1.bmp** and **Bikesgray.jpg** images.

3.5 Segmentation using the Canny detector

Create a new example (**Aula_03_exe_05.py**) that allows, similarly to the previous example, applying the Canny detector to a given image.

Use the function:

```
cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]) →
edges
```

Note that this detector uses hysteresis and needs two threshold values: the larger value (e.g., 100) to determine *"stronger"* contours; the smaller value (e.g., 75) to allow identifying other contours connected to a *"stronger"* one.

Test the developed operations using the **wdg2.bmp**, **lena.jpg, cln1.bmp** and **Bikesgray.jpg** images

Use different threshold values: for instance, 1 and 255; 220 and 225; 1 and 128.

Optional

Perform this operation not on a static image but using the feed of the camera

```
import cv2
capture = cv2.VideoCapture(0)
while (True):
    ret, frame = capture.read()
    cv2.imshow('video', frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
            break

capture.release()
cv2.destroyAllWindows()
```

3.6 Region Segmentation using Flood-Filling

Create a new example (**Aula_03_exe_06.py**) that allows segmenting regions of a given image.

Starting from a **seed pixel**, the **floodFill** function segments a region by spreading the seed value to neighboring pixels with (approximately) the same intensity value.

Use the function

cv2.floodFill(image, mask, seedPoint, newVal[, loDiff[, upDiff[, flags]]]) → retval, rect

to segment the **lena.jpg** image, using as a seed the pixel **(430, 30)** and allowing intensity variations of ±5 regarding the intensity value of the seed pixel.

TASK

Allow the user to interactively select the seed pixel for region segmentation.

Test the interactive region segmentation using the **wdg2.bmp**, **tools_2.png** and **lena.jpg** images.