Lab 3 - Filtering, edge detection

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

3.1 Thresholding

Create a new program (**Aula_03_exe_01.py**) that allows applying Thresholding operations to gray-level images.

Use the corresponding OpenCV function and create a resulting image for each one of the possible operation types: THRESH_BINARY, THRESH_BINARY_INV, THRESH_TRUNC, THRESH_TOZERO and THRESH_TOZERO_INV.

3.2 Averaging Filters

Compile and test the file **Aula_03_exe_02.py**

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

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

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 and compare the results of the successive operations.

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

Use the code of the previous example to 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**

3.3 Median Filters

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

Use the function:

```
dst = cv2.medianBlur(src, ksize[, dst])
```

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

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.

3.4 Gaussian Filters

Create a new example (**Aula_03_exe_04.py**) that allows, similarly to the previous example, applying Gaussian filters to a given image.

Use the function:

```
Dst = cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
```

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

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.

3.5 Computing the image gradient using the Sobel Operator

Compile and test the file **Aula_03_exe_05.py**

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

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

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.

Write additional code to allow applying the (3 × 3) 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.

Optional

Write additional code to apply the (5x5) Sobel operator and evaluate the results with the same (or another) images.

3.6 Canny detector

Create a new example (**Aula_03_exe_06.py**) that allows, similarly to the previous example, applying the Canny detector to a given image.

Use the function:

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

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()
```