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 **OpenCV_ex_13.cpp**

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

```
void blur( InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1),
           int borderType=BORDER_DEFAULT );
```

Tasks

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.

3.2 Filtering Noise

Task

Use the developed code 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 (**OpenCV_ex_14.cpp**) that allows, similarly to the previous example, applying median filters to a given image.

Use the function:

```
void medianBlur( InputArray src, OutputArray dst, int ksize );
```

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

3.4 Filtering Noise

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.

3.5 Gaussian Filters

Create a new example (**OpenCV_ex_15.cpp**) that allows, similarly to the previous example, applying Gaussian filters to a given image.

Use the function:

```
void GaussianBlur( InputArray src, OutputArray dst, Size ksize, double sigmaX,
        double sigmaY=0, int borderType=BORDER_DEFAULT );
```

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

3.6 Filtering Noise

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.

3.8 Computing the image gradient using the Sobel Operator

Compile and test the file **OpenCV_ex_16.cpp**

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

```
void Sobel( InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize=3,
    double scale=1, double delta=0, int borderType=BORDER_DEFAULT );
```

Note the following:

- The resulting image uses a signed, 16-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.

3.8 Segmentation using the Canny detector

Tasks

Create a new example (**OpenCV_ex_17.cpp**) that allows, similarly to the previous example, applying the Canny detector to a given image.

Use the function:

```
void Canny( InputArray image, OutputArray edges, double threshold1, double
        threshold2, int apertureSize=3, bool L2gradient=false );
```

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.

3.9 Region Segmentation using Flood-Filling

Create a new example (**OpenCV_ex_18.cpp**) 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

```
int floodFill( InputOutputArray image, Point seedPoint, Scalar newVal, Rect* rect=0,
               Scalar loDiff=Scalar(), Scalar upDiff=Scalar(), int flags=4 );
```

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.

Tasks

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.