Guião 3

- Filtros: filtragem e remoção de ruído.
- O operador de Sobel: cálculo do gradiente.
- O detector de Canny: segmentação de contornos.
- Segmentação de regiões usando *Flood-Filling*.

3.1 Aplicação de Filtros de Média

Compile e teste o código fornecido no ficheiro **OpenCV_ex_13.cpp**

Analise o código e verifique o modo como é aplicado um filtro de média, usando a função

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

Tarefas

Escreva código adicional que permita:

- Aplicar filtros de média (5 × 5) e (7 × 7) a uma imagem.
- Aplicar sucessivamente (p.ex., 3 vezes) o mesmo filtro à imagem resultante.
- Visualizar os resultados de cada uma das operações efectuadas.

Teste as operações desenvolvidas usando as imagens **Lena_Ruido.png** e **DETI_Ruido.png**

3.2 Filtragem de Ruído

Tarefa

Use a aplicação desenvolvida para analisar os resultados da aplicação de filtros de média a diferentes imagens, comparando os resultados obtidos entre si e com a imagem original.

Use as seguintes imagens de teste:

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

3.3 Aplicação de Filtros de Mediana

Crie um novo programa de exemplo (**OpenCV_ex_14.cpp**) que permita, de modo semelhante ao exemplo anterior, aplicar filtros de mediana.

Use a função

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

Teste as operações desenvolvidas usando as imagens **Lena_Ruido.png** e **DETI_Ruido.png**

3.4 Filtragem de Ruído

Tarefa

Use a aplicação desenvolvida para analisar os resultados da aplicação de filtros de mediana a diferentes imagens, comparando os resultados obtidos entre si, com a imagem original e com os resultados da aplicação de filtros de média.

Use as mesmas imagens de teste que anteriormente.

3.5 Aplicação de Filtros Gaussianos

Crie um novo programa de exemplo (**OpenCV_ex_15.cpp**) que permita, de modo semelhante ao exemplo anterior, aplicar filtros gaussianos.

Use a função

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

Teste as operações desenvolvidas usando as imagens **Lena_Ruido.png** e **DETI_Ruido.png**

3.6 Filtragem de Ruído

Tarefa

Use a aplicação desenvolvida para analisar os resultados da aplicação de filtros gaussianos a diferentes imagens, comparando os resultados obtidos entre si, com a imagem original e com os resultados da aplicação de filtros de média e de mediana.

Use as mesmas imagens de teste que anteriormente.

3.8 Cálculo do gradiente usando o Operador de Sobel

Compile e teste o código fornecido no ficheiro **OpenCV_ex_16.cpp**

Analise o código e verifique o modo como é aplicado o operador de Sobel para calcular as derivadas direcionais de primeira ordem, usando a função

```
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 o seguinte:

- Para a imagem resultante é usada uma representação de 16 bits, com sinal, para cada *pixel*.
- É necessário efectuar posteriormente uma conversão para a representação habitual em níveis de cinzento (8 bits, sem sinal).

Tarefas

Acrescente código que permita aplicar o operador de Sobel (5 × 5).

E combinar as duas derivadas direcionais usando:

$$Destino = GradienteX^2 + GradienteY^2$$

em que *GradienteX* e *GradienteY* representam as derivadas direcionais calculadas com o operador de Sobel.

Teste as operações desenvolvidas usando as imagens **wdg2.bmp**, **lena.jpg, cln1.bmp** e **Bikesgray.jpg**

3.8 Segmentação usando o detector de Canny

Tarefas

Crie um novo programa de exemplo (**OpenCV_ex_17.cpp**) que permita, de modo semelhante ao exemplo anterior, aplicar o detector de Canny.

Use a função

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

Note que este detector utiliza histerese, pelo que precisa de dois *thresholds*: o de maior valor para determinar contornos fortes (por exemplo 100), sendo considerados depois todos os contornos maiores que o valor menor (por exemplo 75), mas ligados a um contorno forte.

Teste as operações desenvolvidas usando as imagens **wdg2.bmp**, **lena.jpg, cln1.bmp** e **Bikesgray.jpg**

Use diferentes valores para os *thresholds*: por exemplo, 1 e 255; 220 e 225; 1 e 128.

3.9 Segmentação de regiões usando *Flood-Filling*

Crie um novo programa de exemplo (**OpenCV_ex_18.cpp**) que permita efectuar a segmentação de regiões.

A função *floodFill* permite segmentar uma região a partir de um *pixel* cuja informação vai ser propagada pelos *pixels* vizinhos com (aproximadamente) o mesmo valor de intensidade.

Use a função

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

para segmentar a imagem **lena.jpg**, usando como semente o *pixel* (430,30) e permitindo variações de 5 unidades relativamente ao valor de intensidade da semente.

Tarefas

Melhore o exemplo para que a selecção de um *pixel* da imagem provoque a segmentação da região a partir desse mesmo *pixel*.

Teste esta segmentação interactiva usando as imagens **wdg2.bmp**, **tools_2.png** e **lena.jpg**.