Nota: Algunas de las imágenes que aparecen en esta presentación provienen del libro:
Visión por Computador: fundamentos y métodos.
Arturo de la Escalera Hueso. Prentice Hall.

Intelligent Systems Lab

Sistemas de Percepción

Visión por Computador

Arturo de la Escalera
José María Armingol
Fernando García
David Martín
Abdulla Al-Kaff

uc3m | Universidad Carlos III de Madrid

Intelligent Systems Lab

Reducción de ruido y detección de bordes: Filtrado especial, Sobel y Canny

Reducción de ruido

- Algoritmos de reducción de ruido
  - Media (Blur).
  - Gaussiana.
  - Mediana.

Reducción de ruido

- BLUR (Media)
  - Cada píxel en la imagen filtrada es la media de los píxeles vecinos (incluidos en la ventana) en la imagen original.

blur

Blurs an image using the normalized box filter.

C++: void **blur**(InputArray **src**, OutputArray **dst**, Size **ksize**, Point **anchor**=Point(-1,-1), int **borderType**=BORDER_DEFAULT)

Parameters:
- **src** – input image; it can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – blurring kernel size.
- **anchor** – anchor point; default value `Point(-1,-1)` means that the anchor is at the kernel center.
- **borderType** – border mode used to extrapolate pixels outside of the image.

The function smoothes an image using the kernel:

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

The call `blur(src, dst, ksize, anchor, borderType)` is equivalent to `boxFilter(src, dst, src.type(), anchor, true, borderType)`.

Reducción de ruido

- MEDIAN BLUR
  - El filtro de la mediana reemplaza cada píxel original (sin filtrar) por el de la mediana de los valores de una ventana cuadrada alrededor del píxel original

medianBlur

Blurs an image using the median filter.

C++: void **medianBlur**(InputArray **src**, OutputArray **dst**, int **ksize**)

Parameters:
- **src** – input 1-, 3-, or 4-channel image; when `ksize` is 3 or 5, the image depth should be `CV_8U`, `CV_16U`, or `CV_32F`, for larger aperture sizes, it can only be `CV_8U`.
- **dst** – destination array of the same size and type as `src`.
- **ksize** – aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

The function smoothes an image using the median filter with the `ksize` × `ksize` aperture. Each channel of a multi-channel image is processed independently. In-place operation is supported.

Reducción de ruido

- GAUSSIAN
  - Realiza una convolución de cada píxel original (sin filtrar) con una máscara Gaussiana
  - El parámetro size determina las dimensiones de la ventana del filtro Gaussiano (máscara Gaussiana).
  - SigmaX es el valor de sigma del filtro Gaussiano. En el caso de no especificar el valor de sigma: el valor se calcula automáticamente a partir de las dimensiones de la ventana del filtro, utilizando las siguientes ecuaciones:

$$\sigma_x = \left(\frac{n_x}{2} - 1\right) \cdot 0.30 + 0.80, \quad n_x = \text{window size width} \qquad \sigma_y = \left(\frac{n_y}{2} - 1\right) \cdot 0.30 + 0.80, \quad n_y = \text{window size height}$$

  - También se puede crear un filtro Gaussiano asimétrico, especificando los valores de sigmaX (dirección horizontal) y sigmaY (dirección vertical).

Reducción de ruido

Intelligent Systems Lab

GaussianBlur

Blurs an image using a Gaussian filter.

C++: void **GaussianBlur**(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)

Parameters:

- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from `sigma*` .
- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height` , respectively (see **`getGaussianKernel()`** for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of `ksize`, `sigmaX`, and `sigmaY`.
- **borderType** – pixel extrapolation method (see **`borderInterpolate()`** for details).

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

Reducción de ruido

- e01_noise_reduction.cpp
  - Cargar la imagen.
  - Comprobar que se ha cargado correctamente.
  - Aplicar el filtro (probar diferentes filtros).
  - Mostrar las imágenes.
  - Esperar la pulsación de una tecla.
  - Liberar memoria

Reducción de ruido

```
8  int main(int argc, char* argv[])
9  {
10     // Object
11     Mat original_img, filtered_img;
12
13     // Load image from disk
14     original_img = imread("moon_ri.jpg", CV_LOAD_IMAGE_GRAYSCALE);
15     if (!original_img.data){
16         cout << "error loading image" << endl;
17         return 1;
18     }
19
20     // Noise reduction by average of the neighbours determined by the mask
21     blur(original_img, filtered_img, Size(3, 3));
22
23     // Gaussian sigma values. The mask dimensions must be square (3x3, 9x9, ...)
24     //int sigma_x = 0;
25     //int sigma_y = 0;
26     //GaussianBlur(original_img, filtered_img, Size(9, 9), sigma_x, sigma_y);
27
28     // Median size parameter must be odd and possitive (1, 3, 5, ...)
29     //int size = 5;
30     //medianBlur(original_img, filtered_img, size);
31
32     // Windows for all the images
33     namedWindow("Original picture", CV_WINDOW_AUTOSIZE);
34     namedWindow("Filtered picture", CV_WINDOW_AUTOSIZE);
35
36     // Show image in the name of the window
37     imshow("Original picture", original_img);
38     imshow("Filtered picture", filtered_img);
```

```
40     // Function for show the image in ms.
41     waitKey(0);
42
43     // Free memory
44     original_img.release();
45     filtered_img.release();
46     destroyAllWindows();
47     // End of the program
48     return 0;
49 }
```

Operador Sobel

Intelligent Systems Lab

Gx

-1	0	1
-2	0	2
-1	0	1

Gy

-1	-2	-1
0	0	0
1	2	1

Sobel

Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator.

C++: 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)

Parameters:

- **src** – input image.
- **dst** – output image of the same size and the same number of channels as `src` .
- **ddepth** –

  output image depth; the following combinations of `src.depth()` and `ddepth` are supported:

  - `src.depth()` = `CV_8U`, `ddepth` = -1/`CV_16S`/`CV_32F`/`CV_64F`
  - `src.depth()` = `CV_16U`/`CV_16S`, `ddepth` = -1/`CV_32F`/`CV_64F`
  - `src.depth()` = `CV_32F`, `ddepth` = -1/`CV_32F`/`CV_64F`
  - `src.depth()` = `CV_64F`, `ddepth` = -1/`CV_64F`

  when `ddepth=-1`, the destination image will have the same depth as the source; in the case of 8-bit input images it will result in truncated derivatives.

Operador Sobel

Intelligent Systems Lab

Sobel

Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator.

C++: 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)

- **xorder** – order of the derivative x.
- **yorder** – order of the derivative y.
- **ksize** – size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
- **scale** – optional scale factor for the computed derivative values; by default, no scaling is applied (see **getDerivKernels()** for details).
- **delta** – optional delta value that is added to the results prior to storing them in `dst`.
- **borderType** – pixel extrapolation method (see **borderInterpolate()** for details).

Operador Sobel

- Si la imagen original es 8-bit, entonces la imagen filtrada debe ser del tipo “16S“ para evitar desbordamiento de memoria.
- La imagen filtrada debe ser normalizada y convertida a 8bits para mostrarla. Esto sucede porque la profundidad es diferente de la imagen original.

Operador Sobel

Imagen filtrada → 16 signed bit (para evitar *overflow*)

```
Mat filtered_img( original_img.size() , CV_16SC1)
Mat show_filtered_img( original_img.size(), CV8UC1)

sobel(original_img, filtered_img, 1, 0, 3)
filtered_img.convertTo(show_filtered_img, CV_8UC1)
```

Operador Sobel

- e02_sobel.cpp
  - Cargar la imagen.
  - Comprobar que la imagen ha sido cargada.
  - Crear Mat con 16 bits con signo (CV_16SC1) para almacenar la imagen filtrada.
  - Aplicar el filtro de Sobel.
  - Mostrar las imágenes.
  - Esperar la pulsación de una tecla.
  - Liberar memoria.
  - Finalizar el programa.

Operador Sobel

```
10  int main(int argc, char* argv[])
11  {
12      // Objects
13      Mat original_img;
14
15      // Load image from disk
16      original_img = imread("lena.jpg", CV_LOAD_IMAGE_GRAYSCALE);
17      if (!original_img.data){
18          cout << "error loading image" << endl;
19          return 1;
20      }
21
22      Mat filtered_img(original_img.size(), CV_16SC1);
23      Mat show_filtered_img_X(original_img.size(), CV_8UC1);
24      Mat show_filtered_img_Y(original_img.size(), CV_8UC1);
25
```

Operador Sobel

```
27      // Sobel edge detector:
28      //  Gx first order
29      //  Gy none
30      //  Mask 3x3
31      Sobel(original_img, filtered_img, CV_16SC1, 1, 0, 3);
32      // Convert to 8 unsigned bit
33      filtered_img.convertTo(show_filtered_img_X, CV_8UC1);
34
35      // Sobel edge detector:
36      //  Gx first order
37      //  Gy none
38      //  Mask 3x3
39      Sobel(original_img, filtered_img, CV_16SC1, 0, 1, 3);
40      // Convert to 8 unsigned bit
41      filtered_img.convertTo(show_filtered_img_Y, CV_8UC1);
```

Operador Sobel

```
42      // Windows for all the images
43      namedWindow("Original picture", CV_WINDOW_AUTOSIZE);
44      namedWindow("Filtered picture X", CV_WINDOW_AUTOSIZE);
45      namedWindow("Filtered picture Y", CV_WINDOW_AUTOSIZE);
46
47      // Show image in the name of the window
48      imshow("Original picture", original_img);
49      imshow("Filtered picture X", show_filtered_img_X);
50      imshow("Filtered picture Y", show_filtered_img_Y);
51
52      // Function for show the image in ms.
53      waitKey(0);
54
55      // Free memory
56      original_img.release();
57      filtered_img.release();
58      show_filtered_img_X.release();
59      show_filtered_img_Y.release();
60
61      destroyAllWindows();
62      // End of the program
63      return 0;
64  }
```

Canny

Canny

Finds edges in an image using the [Canny86] algorithm.

C++: void **Canny**(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)

Parameters:

- **image** – single-channel 8-bit input image.
- **edges** – output edge map; it has the same size and type as `image` .
- **threshold1** – first threshold for the hysteresis procedure.
- **threshold2** – second threshold for the hysteresis procedure.
- **apertureSize** – aperture size for the **`Sobel()`** operator.
- **L2gradient** – a flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (`L2gradient=true`), or whether the default L_1 norm $= |dI/dx| + |dI/dy|$ is enough (`L2gradient=false`).

Canny recomienda utilizar un ratio, **highThresh: lowThresh**, entre: 2:1 o 3:1, pero se pueden utilizar ratios diferentes, 5:1 ó 3:2

Canny

```
10  int main(int argc, char* argv[])
11  {
12      // Objects
13      Mat original_img;
14      Mat filtered_img(original_img.size(), original_img.type());
15
16      // Load image from disk
17      original_img = imread("googleCar.jpg", CV_LOAD_IMAGE_GRAYSCALE);
18      if (!original_img.data){
19          cout << "error loading image" << endl;
20          return 1;
21      }
22
23      /// Reduce noise with a kernel 3x3
24      //blur(original_img, original_img, Size(3, 3));
25
26      // Canny: 2:1 3x3
27      int low_threshold = 50;
28      int high_threshold = low_threshold * 2;
29      int kernel_size = 3;
30      Canny(original_img, filtered_img, low_threshold, high_threshold, kernel_size);
```

Canny

```
32      // Windows for all the images
33      namedWindow("Original picture", CV_WINDOW_AUTOSIZE);
34      namedWindow("Filtered picture", CV_WINDOW_AUTOSIZE);
35
36      // Show image in the name of the window
37      imshow("Original picture", original_img);
38      imshow("Filtered picture", filtered_img);
39      // Function for show the image in ms.
40      waitKey(0);
41
42      // Free memory
43      original_img.release();
44      filtered_img.release();
45      destroyAllWindows();
46      // End of the program
47      return 0;
48  }
```