Visión Artificial

Práctica 3: Filtrado y detección de contornos

David Martín Gómez
Laboratorio de Sistemas Inteligentes
Universidad Carlos III de Madrid

Índice

- Filtros pre-establecidos
  - blur
  - GaussianBlur
  - medianBlur
  - bilateralFilter
- Convolución
- Detección de contornos
  - Sobel
  - Canny

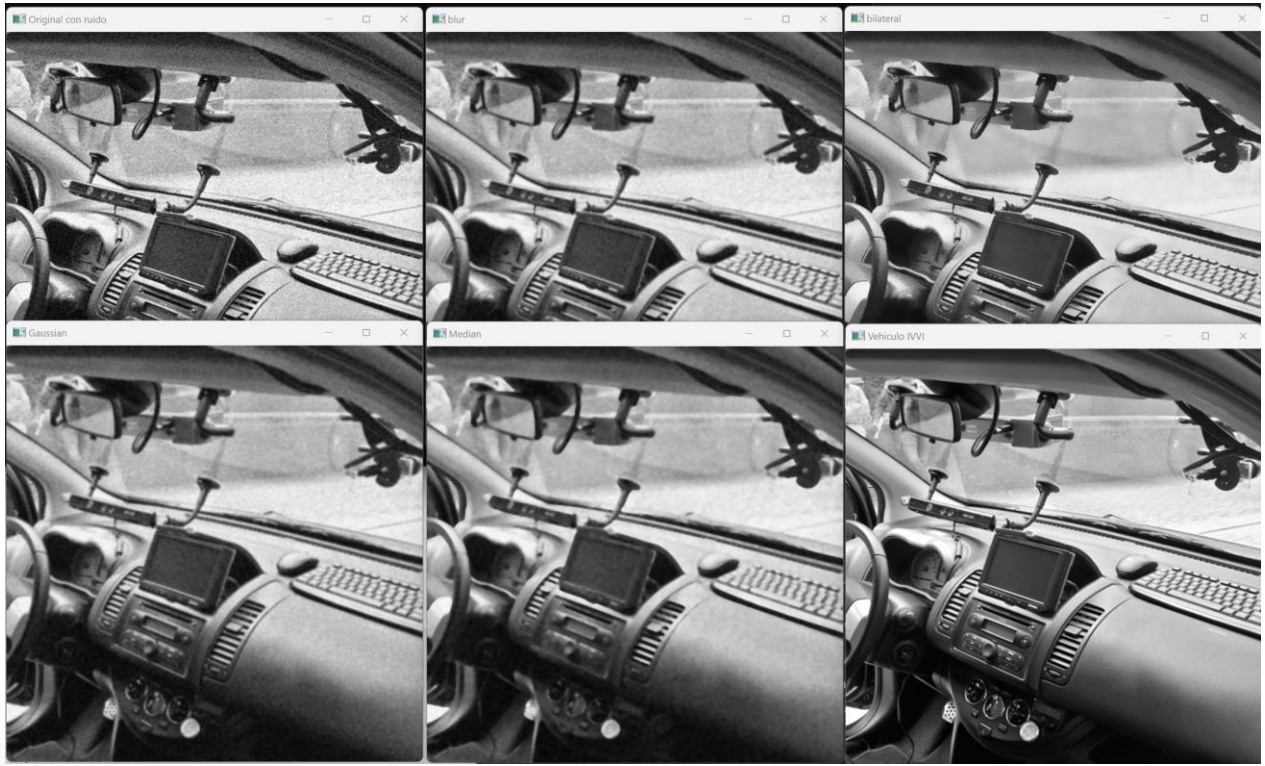
Imágenes

ivvi_512x512_gray.jpg

ivvi_512x512_gray_rg.jpg

ivvi_512x512_gray_ri.jpg

Objetivo


Original con ruido
blur
bilateral
Gaussian
Median
Vehiculo IVVI

Filtros pre-establecidos

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)

Python: cv2.**blur**(src, ksize[, dst[, anchor[, borderType]]]) → dst

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.

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

```
// Filtro de la media
blur(src, imBlr, Size(3, 3), Point(-1, -1), BORDER_DEFAULT);
```

Filtros pre-establecidos

```
// Filtro gausiano
GaussianBlur(src, imGus, Size(5, 5), 0, 0, BORDER_DEFAULT);
```

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.

Filtros pre-establecidos

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.

Filtros pre-establecidos

◆ bilateralFilter()

```
void cv::bilateralFilter ( InputArray  src,
                           OutputArray dst,
                           int         d,
                           double      sigmaColor,
                           double      sigmaSpace,
                           int         borderType = BORDER_DEFAULT
                         )
```

Python:

```
cv.bilateralFilter( src, d, sigmaColor, sigmaSpace[, dst[, borderType]] ) -> dst
```

```
// Filtro bilateral
bilateralFilter(src, imBil, 15, 40, 8);
```

Parameters

src	Source 8-bit or floating-point, 1-channel or 3-channel image.
dst	Destination image of the same size and type as src .
d	Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace.
sigmaColor	Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see sigmaSpace) will be mixed together, resulting in larger areas of semi-equal color.
sigmaSpace	Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace.
borderType	border mode used to extrapolate pixels outside of the image, see **BorderTypes**

Índice

- Filtros pre-establecidos
  - blur
  - GaussianBlur
  - medianBlur
  - bilateralFilter
- Convolución
- Detección de contornos
  - Sobel
  - Canny

Objetivo

Convolución

filter2D

Convolves an image with the kernel.

C++: void **filter2D**(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernel**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)

Python: cv2.**filter2D**(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) → dst

C: void **cvFilter2D**(const CvArr* **src**, CvArr* **dst**, const CvMat* **kernel**, CvPoint **anchor**=cvPoint(-1,-1))

Python: cv.**Filter2D**(src, dst, kernel, anchor=(-1, -1)) → None

Parameters

src – input image.

dst – output image of the same size and the same number of channels as `src`.

ddepth –

desired depth of the destination image; if it is negative, it will be the same as `src.depth()`;

Convolución

```
Mat ImagenFilt(imagen.cols,imagen.rows,CV_8U);
//Mat ImagenFilt;

Mat RelBord(3,3,CV_8S);
RelBord.at<char>(0)=-1;RelBord.at<char>(1)=-1;RelBord.at<char>(2)=-1;
RelBord.at<char>(3)=-1;RelBord.at<char>(4)= 9;RelBord.at<char>(5)=-1;
RelBord.at<char>(6)=-1;RelBord.at<char>(7)=-1;RelBord.at<char>(8)=-1;

filter2D(imagen,ImagenFilt,-1,RelBord);//,Point(-1,-1),0);//,BORDER_DEFAULT);
```

Índice

- Filtros pre-establecidos
  - blur
  - GaussianBlur
  - medianBlur
  - bilateralFilter
- Convolución
- Detección de contornos
  - Sobel
  - Canny

Objetivo

Sobel

Sobel()

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

Python:

```
cv.Sobel( src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]] ) -> dst
```

Sobel

convertScaleAbs()

```
void cv::convertScaleAbs ( InputArray  src,
                           OutputArray dst,
                           double      alpha = 1,
                           double      beta = 0
                         )
```

Python:

```
cv.convertScaleAbs( src[, dst[, alpha[, beta]]] ) -> dst
```

addWeighted()

```
void cv::addWeighted ( InputArray  src1,
                       double      alpha,
                       InputArray  src2,
                       double      beta,
                       double      gamma,
                       OutputArray dst,
                       int         dtype = -1
                     )
```

Python:

```
cv.addWeighted( src1, alpha, src2, beta, gamma[, dst[, dtype]] ) -> dst
```

```
#include <opencv2/core.hpp>
```

Calculates the weighted sum of two arrays.

The function addWeighted calculates the weighted sum of two arrays as follows:

$$\texttt{dst}(I) = \texttt{saturate}(\texttt{src1}(I) * \texttt{alpha} + \texttt{src2}(I) * \texttt{beta} + \texttt{gamma})$$

Sobel

threshold()

double cv::threshold (**InputArray** src,
OutputArray dst,
double thresh,
double maxval,
int type
)

Python:

cv.threshold(src, thresh, maxval, type[, dst]) -> retval, dst

threshold types

Enumerator	
THRESH_BINARY Python: cv.THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC Python: cv.THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO Python: cv.THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_MASK Python: cv.THRESH_MASK	
THRESH_OTSU Python: cv.THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE Python: cv.THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

Canny

Canny() [1/2]

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

Python:

```
cv.Canny( image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]] ) -> edges
cv.Canny( dx, dy, threshold1, threshold2[, edges[, L2gradient]]                  ) -> edges
```

```
#include <opencv2/imgproc.hpp>
```

Finds edges in an image using the Canny algorithm **[43]** .

The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges. See http://en.wikipedia.org/wiki/Canny_edge_detector

Parameters

image	8-bit input image.
edges	output edge map; single channels 8-bit image, which has the same size 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 $= \lvert dI/dx \rvert + \lvert dI/dy \rvert$ is enough (L2gradient=false).