HOMEWORK 3 U1 SMOOTHING RESEARCH

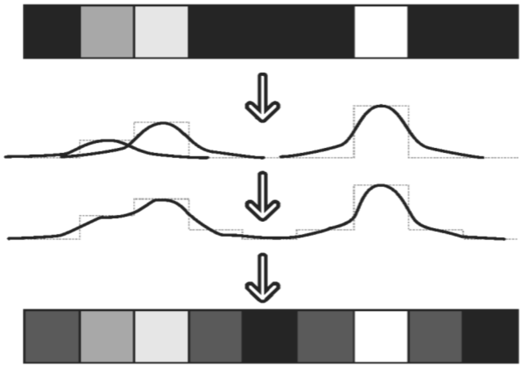
Héctor Mauricio Mendoza Xicoténcatl

Abstract—In this report we documented our research about different types of Smoothing Filters.

Index Terms—Thresholds, Color Segmentation, OpenCv2, Adaptive Threshold, Adaptive Gaussian, Median Filter, Gaussian Filter, Bilateral Filter.

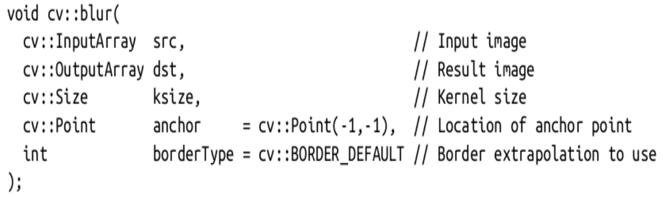
1 SMOOTHING

Smoothing, also called blurring, is a simple and frequently used image-processing operation. There are many reasons for smoothing, but it is often done to reduce noise or camera artifacts.



2 SIMPLE BLUR AND THE BOX FILTERS

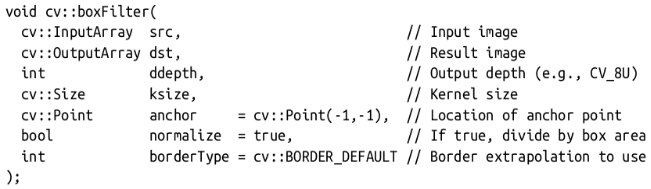
2.1 SIMPLE BLUR



```
void cv::blur(
  cv::InputArray  src,                          // Input image
  cv::OutputArray dst,                          // Result image
  cv::Size        ksize,                        // Kernel size
  cv::Point       anchor     = cv::Point(-1,-1), // Location of anchor point
  int             borderType = cv::BORDER_DEFAULT // Border extrapolation to use
);
```

The simple blur operation is provided by **cv::blur().** Each pixel in the output is the simple mean of all of the pixels in a window (i.e., the kernel), around the corresponding pixel in the input. The argument anchor can be used to specify how the kernel is aligned with the pixel being computed.

2.2 BOX FILTERS

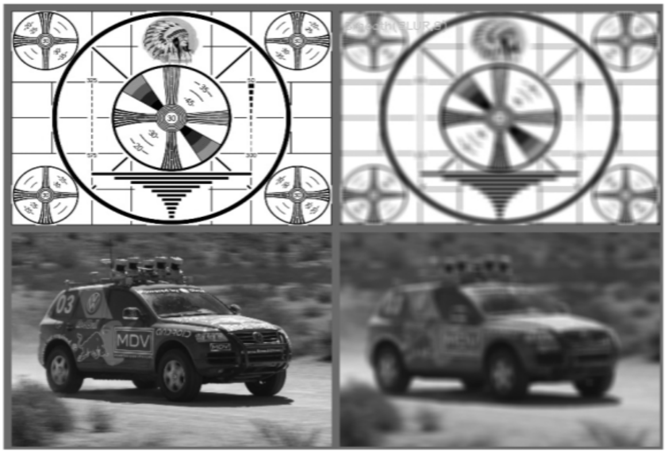


```
void cv::boxFilter(
  cv::InputArray  src,                          // Input image
  cv::OutputArray dst,                          // Result image
  int             ddepth,                       // Output depth (e.g., CV_8U)
  cv::Size        ksize,                        // Kernel size
  cv::Point       anchor     = cv::Point(-1,-1), // Location of anchor point
  bool            normalize  = true,            // If true, divide by box area
  int             borderType = cv::BORDER_DEFAULT // Border extrapolation to use
);
```

The simple blur is a specialized version of the box filter. A box filter is any filter that has a rectangular profile and for which the values $k_(i, j)$ are all equal. In most cases, $k_(i, j) = 1$ for all i, j, or $k_(i, j) = 1/A$, where A is the area of the filter.

The OpenCV function **cv::boxFilter()** is the somewhat more general form of which **cv::blur()** is essentially a special case. The main difference between **cv::boxFilter()** and **cv::blur()** is that the former can be run in an unnormalized mode **(normalize = false)**, and that the depth of the output image **dst** can be controlled.

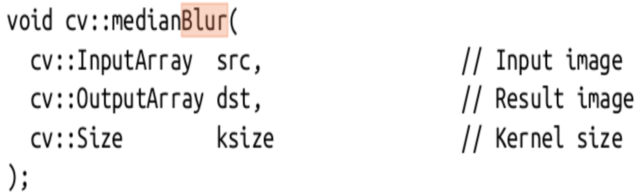
2.3 EXAMPLE



Simple blur and box filters example.

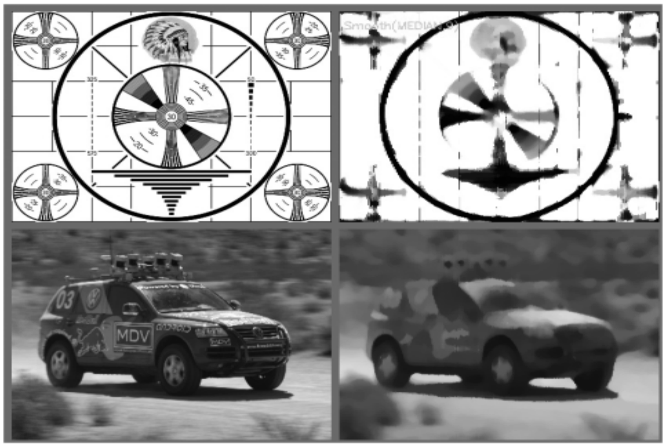
3 MEDIAN FILTER

The median filter replaces each pixel by the median or "middle-valued" pixel (as opposed to the mean pixel) in a rectangular neighborhood around the center pixel.
Median filtering is able to ignore the outliers by selecting the middle points.



```
void cv::medianBlur(
  cv::InputArray  src,                    // Input image
  cv::OutputArray dst,                    // Result image
  cv::Size        ksize                   // Kernel size
);
```

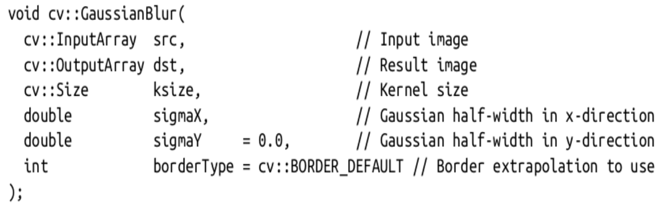
3.1 EXAMPLE



Median filter example.

4 GAUSSIAN FILTER

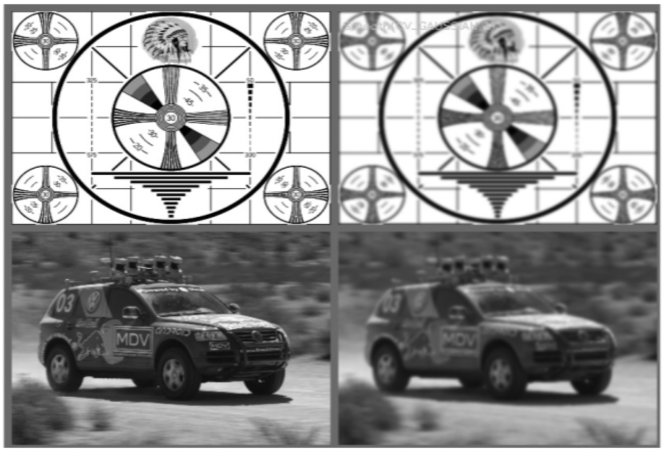
Gaussian filtering involves convolving each point in the input array with a (normalized) Gaussian kernel and then summing to produce the output array.



```
void cv::GaussianBlur(
  cv::InputArray  src,                    // Input image
  cv::OutputArray dst,                    // Result image
  cv::Size        ksize,                  // Kernel size
  double          sigmaX,                 // Gaussian half-width in x-direction
  double          sigmaY     = 0.0,       // Gaussian half-width in y-direction
  int             borderType = cv::BORDER_DEFAULT // Border extrapolation to use
);
```

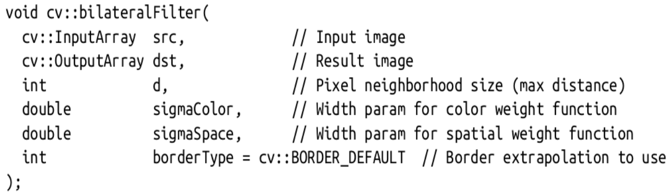
For the Gaussian blur, the parameter ksize gives the width and height of the filter window. The next parameter indicates the sigma value (half width at half max) of the Gaussian kernel in the x-dimension.
The fourth parameter similarly indicates the sigma value in the y-dimension. If you specify only the x value, and set the y value to 0 (its default value), then the y and x values will be taken to be equal. If you set them both to 0 , then the Gaussian's parameters will be automatically determined from the window size.

4.1 EXAMPLE



Gaussian filter example.

5 BILATERAL FILTER

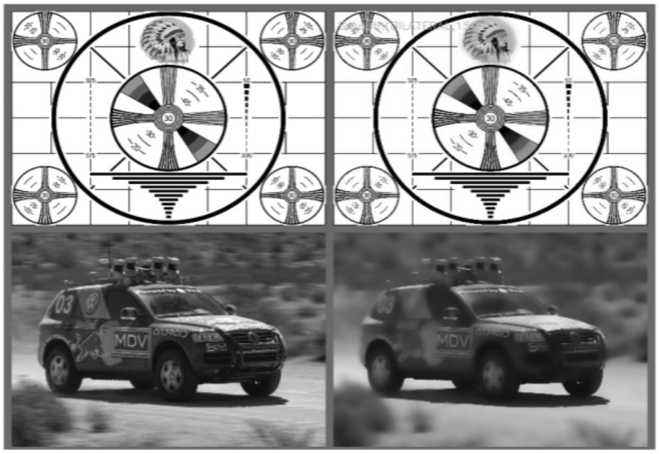


```
void cv::bilateralFilter(
  cv::InputArray  src,            // Input image
  cv::OutputArray dst,            // Result image
  int             d,              // Pixel neighborhood size (max distance)
  double          sigmaColor,     // Width param for color weight function
  double          sigmaSpace,     // Width param for spatial weight function
  int             borderType = cv::BORDER_DEFAULT  // Border extrapolation to use
);
```

Bilateral filtering is one operation from a somewhat larger class of image analysis operators known as edge-preserving smoothing.

Like Gaussian smoothing, bilateral filtering constructs a weighted average of each pixel and its neighboring components. The weighting has two components, the first of which is the same weighting used by Gaussian smoothing. The second component is also a Gaussian weighting but is based not on the spatial distance from the center pixel but rather on the difference in intensity from the center pixel. We can think of bilateral filtering as Gaussian smoothing that weighs similar pixels more highly than less similar ones, keeping high-contrast edges sharp. The effect of this filter is typically to turn an image into what appears to be a watercolor painting of the same scene. This can be useful as an aid to segmenting the image.

5.1 EXAMPLE



Bilateral filter example.

6 CONCLUSIONS

This research gives us the opportunity to learn more about adaptive thresholds. Thus, giving us more tools to use in different types of image processing. I consider these new types of thresholds can help us to solve more complex exercises and problems. I particularly used one of them in the previous activity, the reason was the illumination of the image itself. In this case, the most suitable option was the use of the median and Gaussian threshold.

REFERENCES

[1] A. Kaehler and G. Bradski, Learning OpenCV 3, 1st ed. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.: Kristen Brown, 2020, pp. 261-267.

Héctor Mendoza Héctor Mauricio Mendoza Xicoténcatl is a Computational Robotics Junior student, currently stu-diying Universidad Politécnica de Yucatán.