Marker based localization

Image Filtering

Team members

Niharika Jayanthi Dheeraj Kamath

Under the guidance of
Sanam Shakya

Goals

- To understand what is image filtering
- Learn about various available image filters in python.
- Learn to code for Gaussian, box and median filters.

Theory

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (e.g: noise, edges) from the image resulting in edges being blurred when this is filter is applied. We will discuss three such filters: * Blur filter * Gaussian filter * Median Blur filter

Blur

Working Principle

Also known as Averaging, this technique uses a normalized box filter to convolve images. The low-pass filter kernel is positioned in the image such that it is a

subset of the image. The pixel lying in the center of the kernel will get its value calculated. All the pixels in the kernel area will be used to find the average, which is assigned as the new value of the central pixel. This is an example of a normalized box filter-

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 1: Blur Kernel

The following function is used to blur an image:

```
cv2.blur( src, ksize[, dst[, anchor[, borderType]]])
```

Parameters:

- **src** - Input image that can have any number of input channels. The depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
- **ksize** - Size of kernel used for blurring
- **dst** - Output image of the same size and type as src.
- **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 the image.

Gaussian Blur

Working Principle

In this method, a Gaussian kernel is used for filtering. Gaussian filter is a low pass filter. It is done by convolving each point in the input array with a Gaussian kernel and then summing them all up to produce the output array.
Gaussian filtering is linear, meaning it replaces each pixel by a linear combination of its neighbors (in this case with weights specified by a Gaussian matrix). It is also local, meaning it produces output pixel values based only upon the pixel values in its neighborhood as determined by the convolution kernel. We should

specify the width and height of the kernel which should be positive and odd. Here is an example of 5x5 Gaussian kernel:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 2: Gaussian Kernel

The value of the pixel obtained on evaluating all the pixel values of the neighborhood using the Gaussian function may not be present in the image itself. Hence it results in some blurring. It is also not edge preserving.
A 2D Gaussian function is as given below:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Gaussian kernel coefficients are sampled using the above function. For perfect filtering medianblur() is preferred.

The Gaussian blur can be applied to the images by the following function:

`cv2.GaussianBlur( src, ksize, sigmaX, dst, sigmaY, borderType)`

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.
* **ksize** - Gaussian kernel size.
* **sigmaX** - Gaussian kernel standard deviation in X direction.
* **dst** - Output image of the same size and type as src.
* **sigmaY** - Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX.
* **borderType** - Pixel extrapolation method

Median Blur

In this type, the central pixel's value is calculated by finding the median of the pixel values in the kernel areas. Median refers to the central value in a given set of values. Median blur differs from

Gaussian and box filters in one aspect- The central element is always replaced by a pixel value that is present in the image. In the other filters, filtered value for central element may be a value that is not present in the original image. This property of Median Blur reduces noise effectively.

The function used for median blur is-

```
cv2.medianBlur(src, ksize[, dst])
```

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.
- **ksize** - Aperture linear size; it must be odd and greater than 1.
- **dst** - Destination array of the same size and type as src.

Applications:

- These filters are used in edge detection.
- They effective in smoothening images.
- They can remove noise(like salt and pepper noise) from an image.

Code

Blur

```
#Imports
import cv2
from matplotlib import pyplot as plt

#Read image
img = cv2.imread('example.jpg')

#Blur the image
```

```
  blur = cv2.blur(img, (5,5))

  #Plot both images and compare
  plt.figure(0)

  plt.subplolt(121) #In a grid of 1 row and 2 columns, use first subplot
  plt.imshow(img), plt.title('Original')
  plt.xticks([]), plt.yticks([]) #Remove ticks

  plt.subplot(122) #Use second subplot
  plt.imshow(blur), plt.title('Blurred')
  plt.xticks([]), plt.yticks([])

  plt.show()
```

Consider the following image:

On applying blur, it looks like this-

Gaussian Blur

```
#Import OpenCV and Numpy
import numpy
import cv2

#Read the image
img = cv2.imread('pepper.png')

'''
OpenCV represents RGB images as multi-dimensional NumPy arrays...but
 in reverse order! This means that images are actually represented in
 BGR order rather than RGB!
'''
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)   #Coverting from bgr to rgb

# Do the processing
blur = cv2.GaussianBlur(img,(5,5),0)

plt.figure(0)       # Create a new window
```

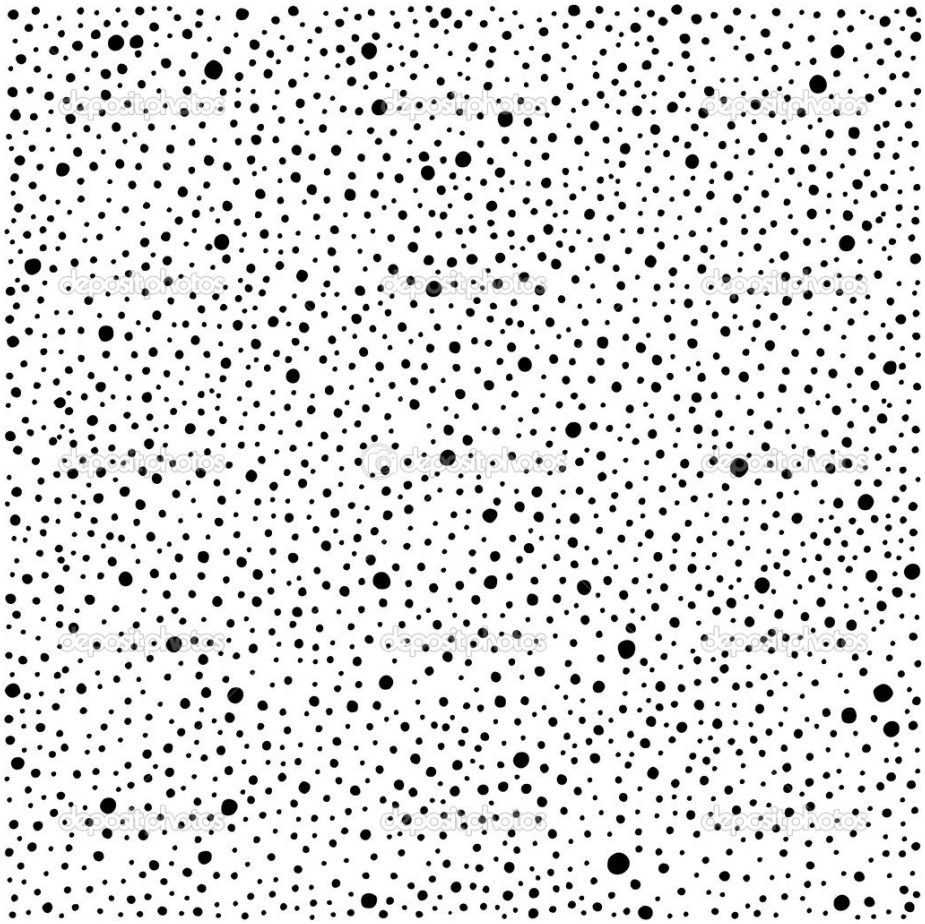

Figure 3: Example image

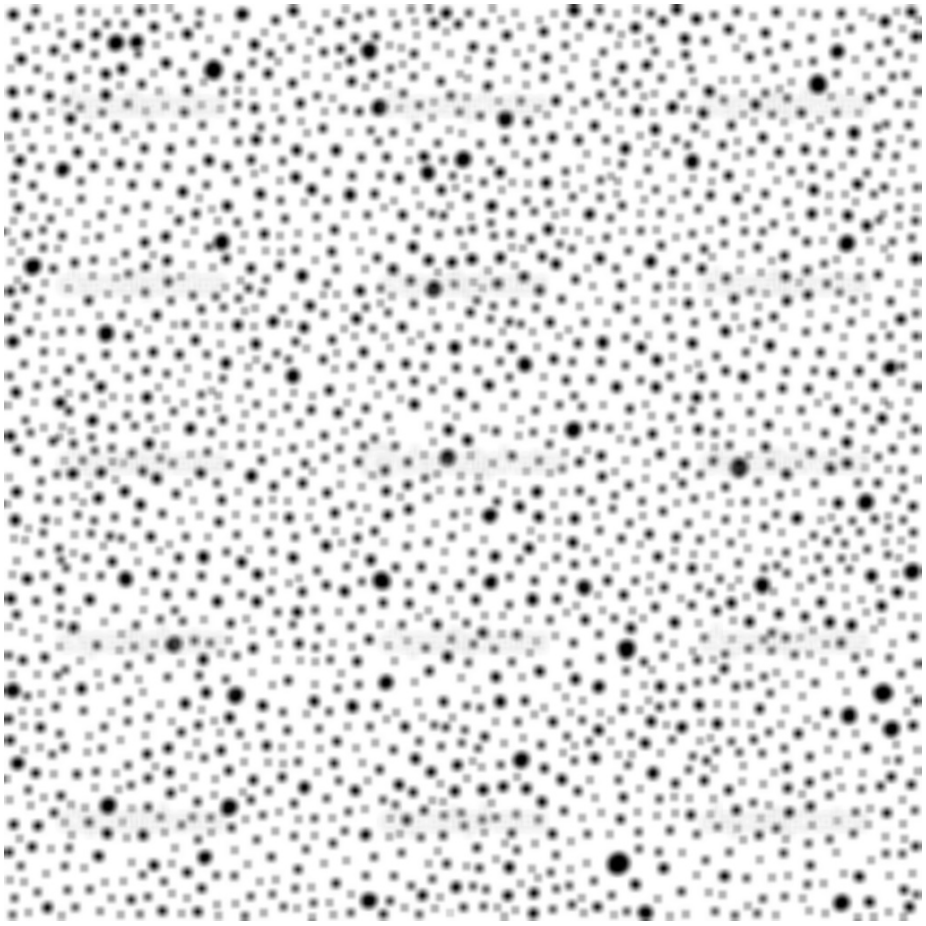

Figure 4: blur

```
plt.subplot(121),plt.imshow(img), plt.title('Original')
plt.xticks([]), plt.yticks([])      #Removing ticks

plt.subplot(122),plt.imshow(blur), plt.title('Gaussian')
plt.xticks([]), plt.yticks([])

plt.show()          #Display the window
```

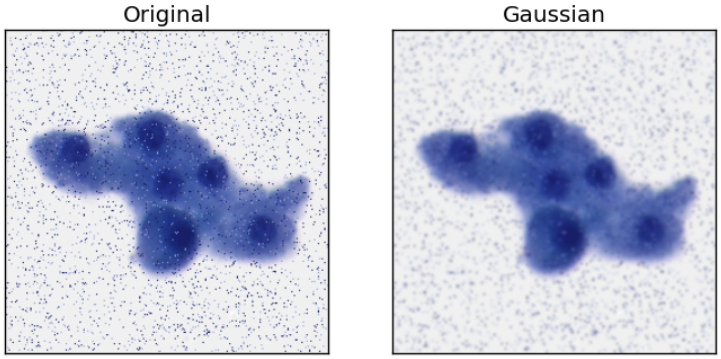

Figure 5: Gauss Image

Median Blur

```
#Imports
import cv2
from matplotlib import pyplot as plt

#Read image
img = cv2.imread('example.jpg')
```

```
#Apply median blur
 mBlur = cv2.medianBlur(img, 5)

#Plot both images and compare
plt.figure(0)

plt.subplolt(121) #In a grid of 1 row and 2 columns, use first subplot
plt.imshow(img), plt.title('Original')
plt.xticks([]), plt.yticks([]) #Remove ticks

plt.subplot(122) #Use second subplot
plt.imshow(mBlur), plt.title('Blurred')
plt.xticks([]), plt.yticks([])

plt.show()
```

Consider the following image-

Figure 6: Example image

On apply median blur, it looks like this-

Figure 7: Median Image

Resources

- Image filtering - opencv and python tutorials
- Filtering - opencv docs

Exercises