Blob Detector

March 20, 2023

1 Introduction

Implement the convolution of a Laplacian of a Gaussian blob (LoG) detector with an image and apply it to 3-4 images of your choice from the dataset (possibly from different thematic classes). Do not use library functions for implementing the convolution or to generate the LoG filter. Implement your own and show the code (the interesting bits at least)! The function you implement should be able to run the LoG for different choices of the scale parameter, which is passed as an input argument. Show the results of your code on the 3-4 example images, for different choices of the scale parameter (sigma).

The code of the module "blob_utils" and this notebook executed can be found on GitHub https://github.com/matteotolloso/ispr or Google Colab ?????

Useful libraries, "blob utils" that contains the core is writted by me.

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import latexify
import cv2
import blob_utils
```

2 Code explanation

2.1 LoG function

Definition of the Laplacian of Gaussian (LoG) function:

[ ]:
```
@latexify.function(use_math_symbols=True)
def LoG(x, y, sigma):
    pi = np.pi
    return ((- 1 / (pi * sigma**4) ) * (1 - (x**2 + y**2) / (2 * sigma**2) ) )␣
↪* np.exp(-( (x**2 + y**2) / (2 * sigma**2)))
LoG
```

[ ]:

$$\pi = np.pi$$
$$\mathrm{LoG}(x, y, \sigma) = \frac{-1}{\pi \sigma^4} \left(1 - \frac{x^2+y^2}{2\sigma^2}\right) \exp\left(-\left(\frac{x^2+y^2}{2\sigma^2}\right)\right)$$

2.2 Kernel building

Building the kernel sampling the function. The kernel must be big enough to cover both the positive and negative part of the function, otherwise we will not be able to detect the blobs.

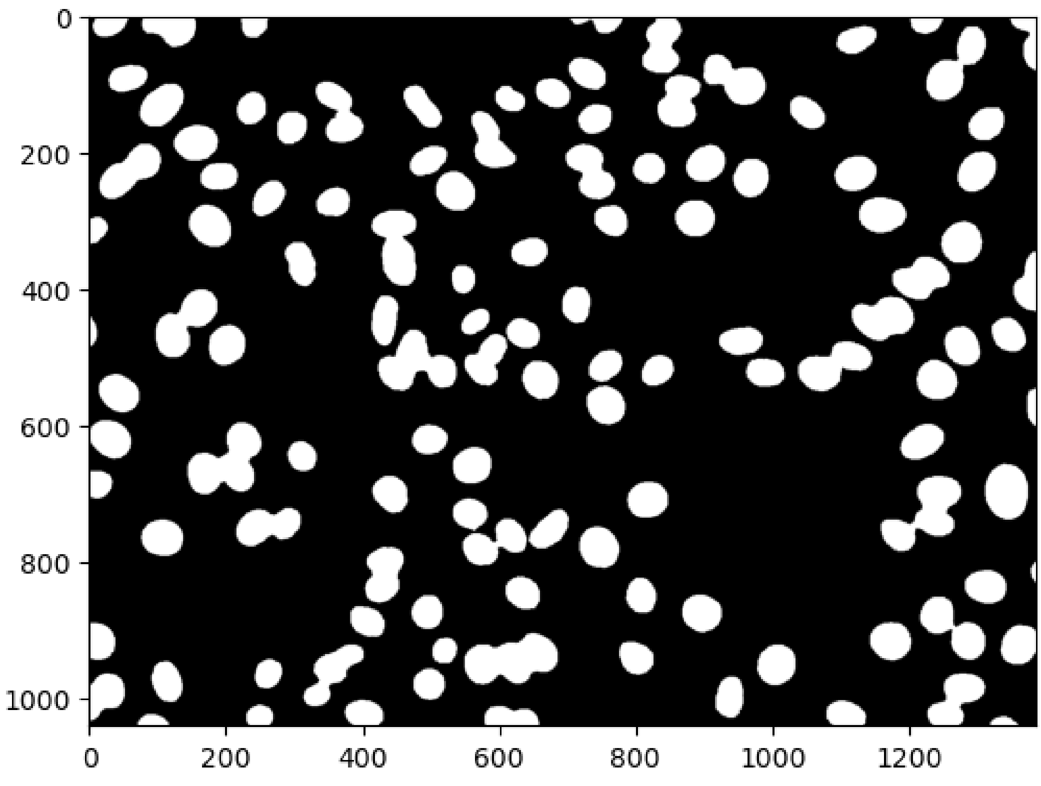
[ ]:
```
KERNEL_SIZE = 51
SIGMA = 14
PATH = "easy.png"
kernel = blob_utils.get_kernel(fun=LoG, size=KERNEL_SIZE, sigma=SIGMA)
```

2.3 Image preprocessing

Transform the image to grayscale

[ ]:
```
rgb_image = cv2.imread(PATH)
gray_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_image, cmap="gray")
```

[ ]: <matplotlib.image.AxesImage at 0x7f4368dc9b50>



Scaling of the image values between -1 and 1 in order to more easily interpret subsequent results.

[ ]:
```
gray_image = np.interp(
    gray_image,
    (gray_image.min(), gray_image.max()),
    (-1, 1)
)
```

2.4 Convolution

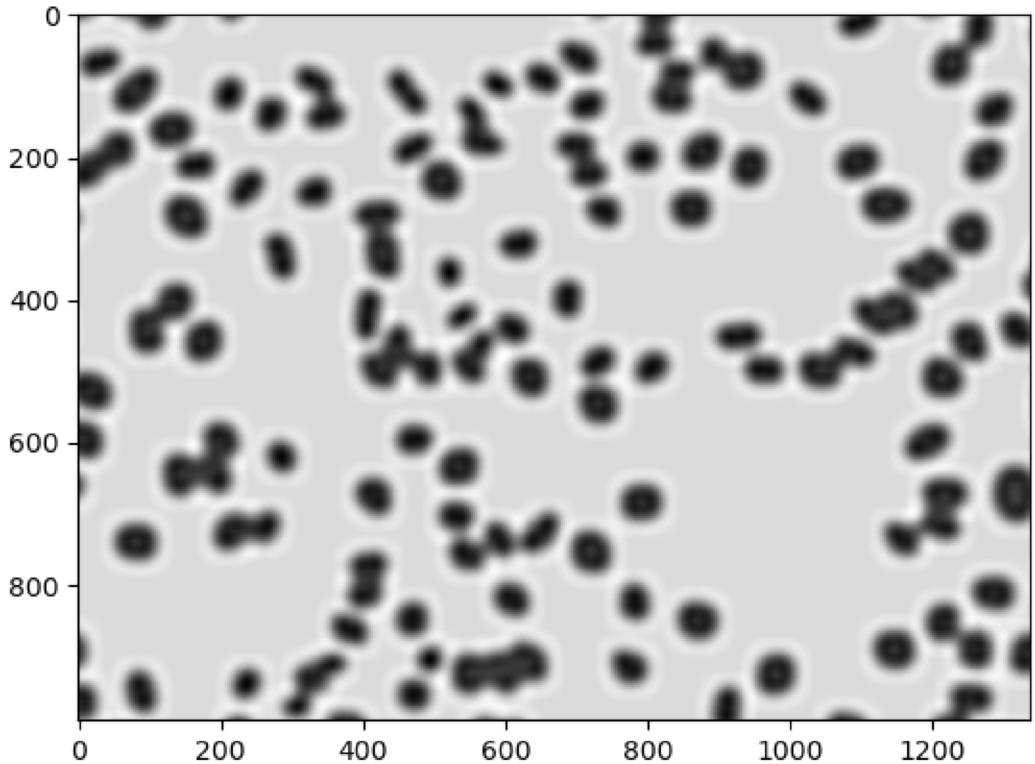
Convolution step. A pixel in the convolved matrix is the value of the sum of the element-wise matrix multiplication between the submatrix of the original image centered in that pixel and the kernel.

If the image has a blob of the kind "dark inside, bright ouside" the center of that blob in the convolved matrix will have an high value since the dark part (negative) is multiplied by the center of the LoG (negative) and the bright part (positive) is multiplied by the positive circular part of the LoG.

For the same reason the blobs of the kind "bright inside, dark outside" will be identified by a low value in the convolved matrix.

[ ]:
```
convolved_image = blob_utils.convolve_image(
    image=gray_image,
    kernel=kernel
)
plt.imshow(convolved_image, cmap="gray")
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f435e5c29d0>
```



As you can see, the black backgrond (value -1) became gray since the high response with the negative center of the kernel, and for the same season, the white circles became light black. In the center of the circles there are some white pixels.

Theoretically, if the blob rasius inducted by sigma perfectly matches the size of the blobs in the image, the value at the center of the blobs in the colvolved image should be greater in absolute value compared to the value of the background, but since it's difficult to match the correct sigma and in addiction the blobs are never perfectly cercular, could happen that the if we use the global minima and maxima ad blob center nothing significant is captured.

Another issue is that a group of pixels in the center of a blob could have an high value (with respect to the rest of the convolved image), and we have to choose wich one will be the center of the blob.

2.5 Finding the blob centers

To address these issues, we are look for local minima and maxima, in particular strictly maxima and minima i.e. a point of the convolved image is the center of a blob if it is strictly greater (minor) that all other points in the blob radius.

In the function there is also the possibility to set a treshold based on the percentile in order to decide the number of blobs that we want to see.

[ ]:
```
blob_radius = int (np.sqrt(2) * SIGMA) + 1
centers = blob_utils.get_centers(
```

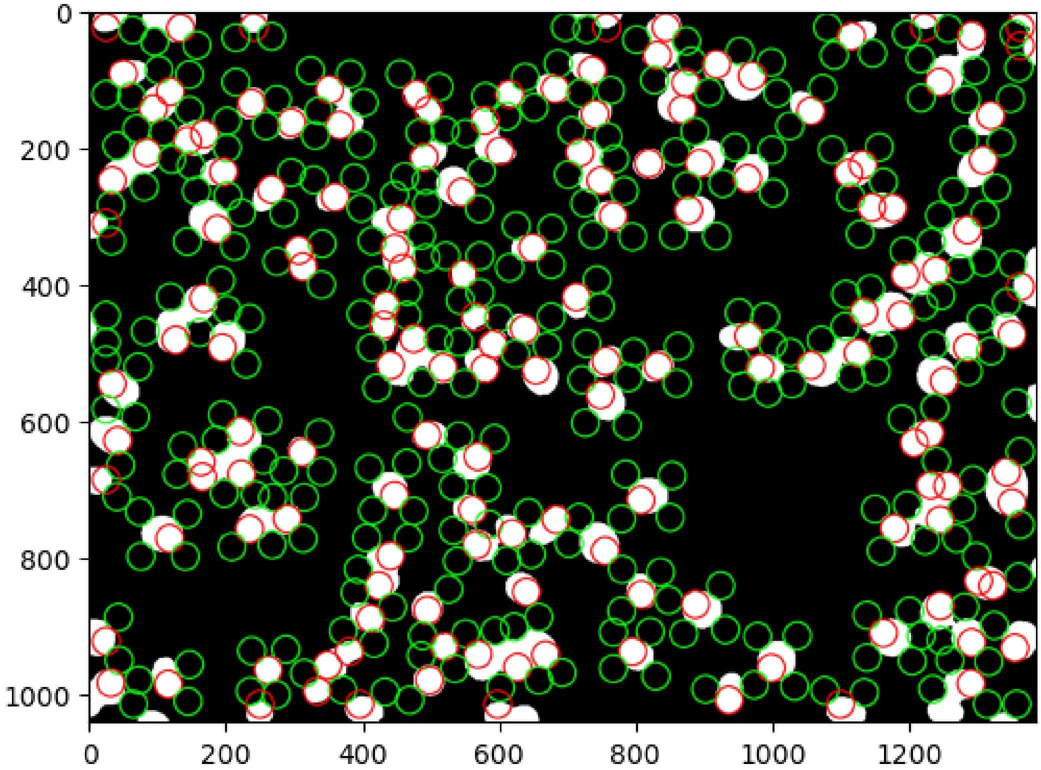
```
    convolved_image,
    blob_radius=blob_radius,
    kernel_size=kernel.shape[0],
    percentile=np.percentile(convolved_image, [40, 60])
)
```

2.6 Final result drawing

The blobs given by a local mimium are circled in green, while the ones given by a local maximum are red.

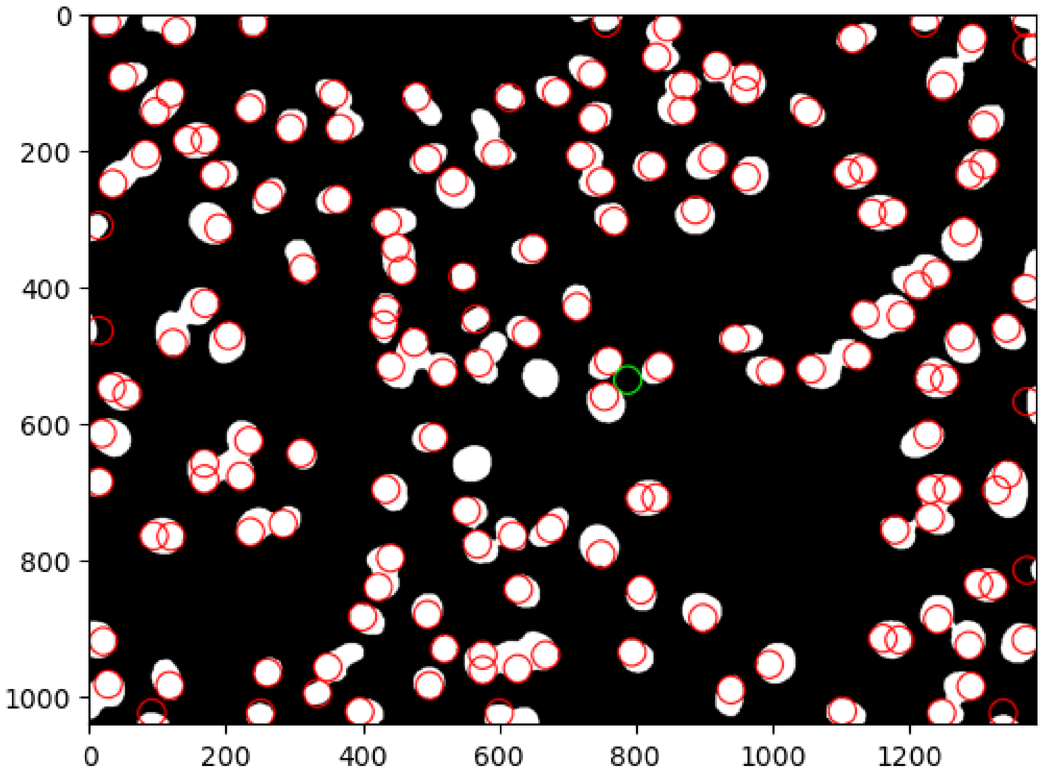
[ ]:
```
rgb_image = cv2.imread(PATH)
for (i, j , _, tipo) in centers:
    if tipo=="min":
        color = (255, 0, 0)
    else:
        color = (0, 255, 0)
    cv2.circle(
        rgb_image,
        (j, i),
        blob_radius,
        color,
        thickness=2,
        lineType=2
    )

plt.imshow(rgb_image, cmap="gray")
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f4368cb7430>
```



There are a lot of wrong blobs because the treshold on the percentile is symmetric for black and white blobs (lower that 40 and higher than 60), while in this image we only have white blobs, this behaviour can be corrected increasing the treshold for the black blobs (as in the following example) or setting a global treshold.

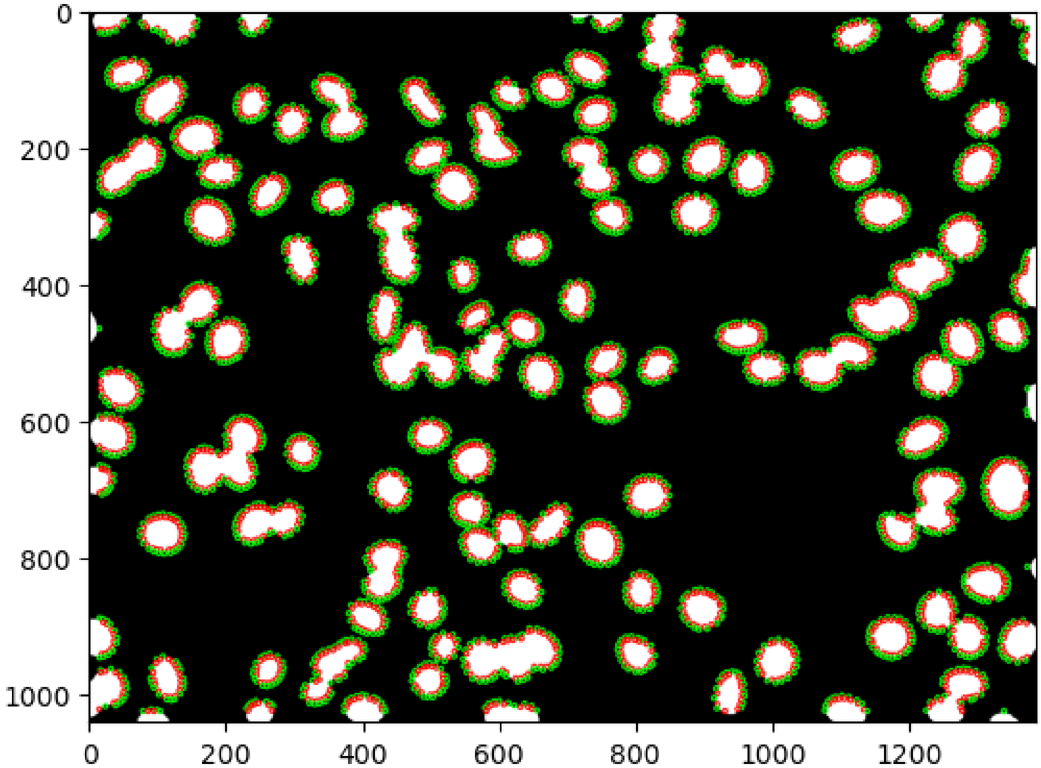
[ ]:
```
blob_utils.full_pipeline(
    path="easy.png",
    kernel_size = 31,
    sigma = 14,
    percentile = (30, 99.99999), # basically no positive blobs
    line_tickness=2
)
```



2.7 Edge detector

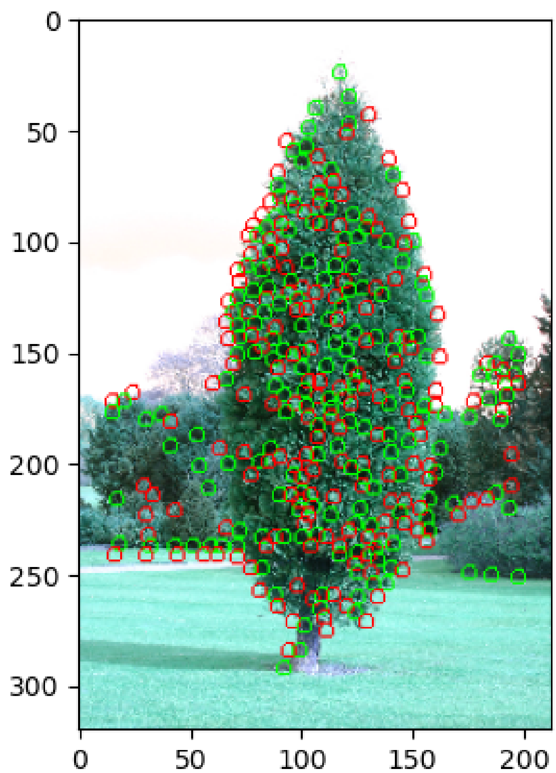
In some particular cases, a blob detector implemented in this way can also be used to perform edge detection setting a small sigma. That is because the internal part of the blob is seen as a flat region, while the border regions of the blob will have an high response in the convolution phase. An example below.

[ ]:
```
blob_utils.full_pipeline(
    path="easy.png",
    kernel_size = 31,
    sigma = 2,
    percentile = (30, 70),
    line_tickness=2
)
```

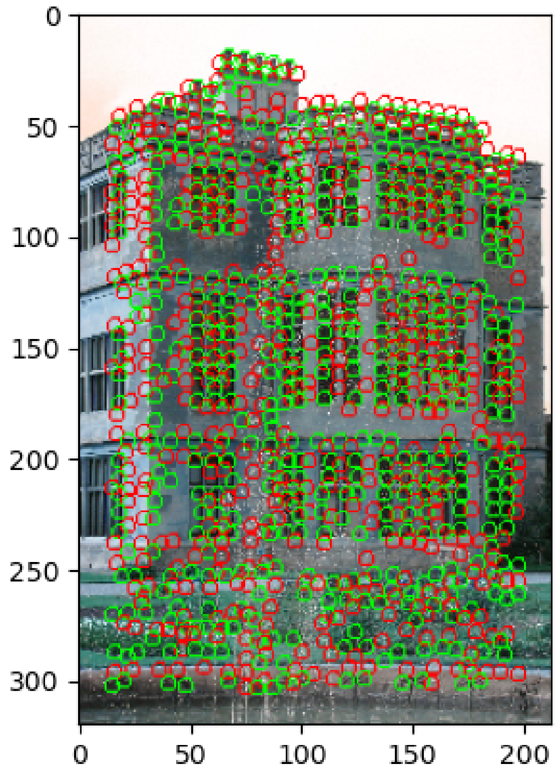


3 Examples in the database

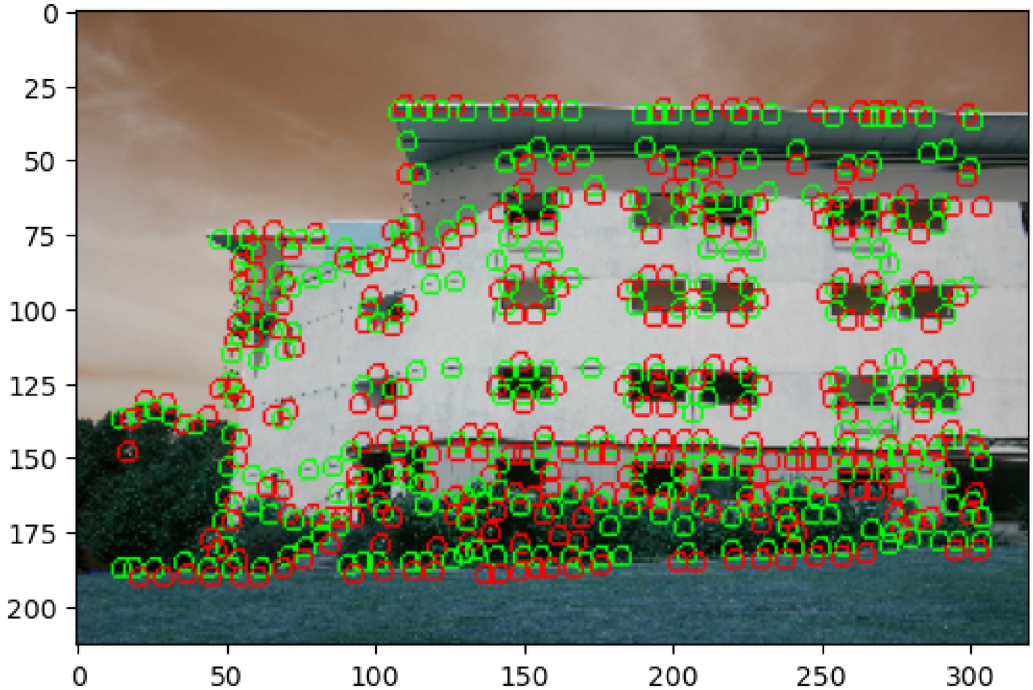
[ ]:
```
blob_utils.full_pipeline(
    path="./2_21_s.bmp",
    kernel_size = 31,
    sigma = 2,
    percentile = (3, 97),
    line_tickness=1
)
```



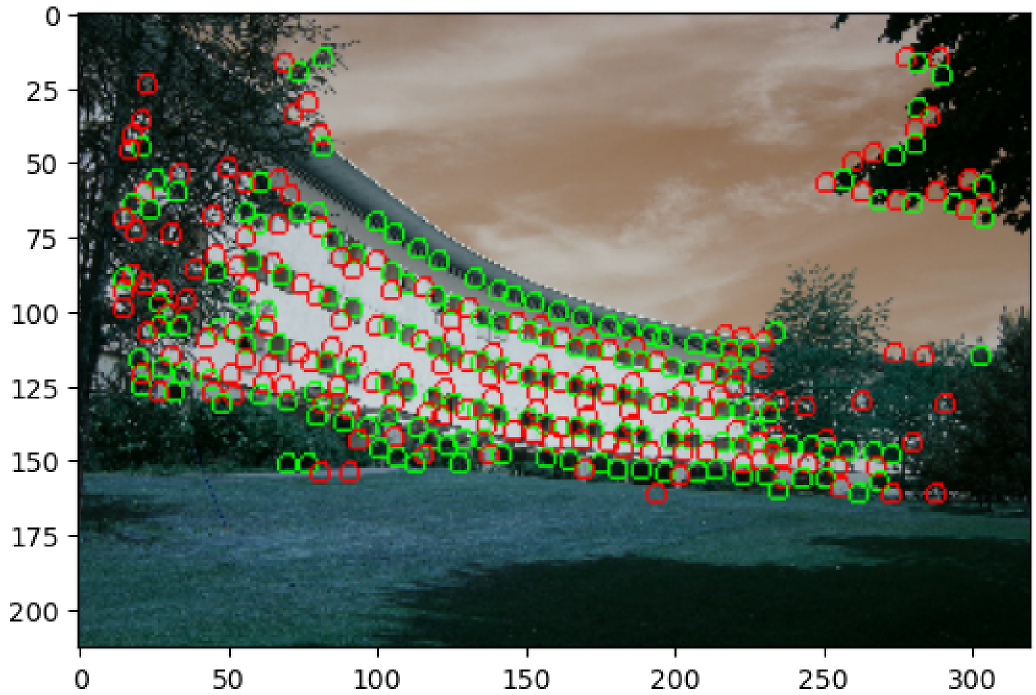
[ ]:
```
blob_utils.full_pipeline(
    path="./3_2_s.bmp",
    kernel_size = 31,
    sigma = 2,
    percentile = (15, 85),
    line_tickness=1
)
```




```
[ ]: blob_utils.full_pipeline(
         path="./3_24_s.bmp",
         kernel_size = 31,
         sigma = 1.5,
         percentile = (10, 90),
         line_tickness=1
     )
```



[ ]:
```
blob_utils.full_pipeline(
    path="./3_25_s.bmp",
    kernel_size = 31,
    sigma = 2,
    percentile = (5, 95),
    line_tickness=1
)
```



4 Conclusions

The results are good but require some manual adjustmens of the sigma and the tresholds on the percentile (but in most of the cases the tresholds can by symmetric)