1 System Overview

The following steps outline the high-level implementation and are referenced in section 2 of this report.

1. Compute an octave in scale-space.
2. Compute the squared Difference of Gaussians (DoG) using the octave computed in step 1.
3. Compute the location of the maxima of the octave of squared Difference of Gaussians computed in step 2.
4. Down-sample the third image from the bottom of the octave, which has been effectively blurred by 2σ, by a factor of 2 to obtain the first layer in the next scale-space octave. This approach is the same approach used by David Lowe, and this is highlighted in the last paragraph before section 3.1 in [1].
5. Go back to step 1 and repeat until the desired number of octaves has been generated.
6. Display circles around the centers of the detected blobs.

Figure 1.1 below shows the hierarchy of functions used, where the arrow indicates that the upper function uses/depends on the lower function.

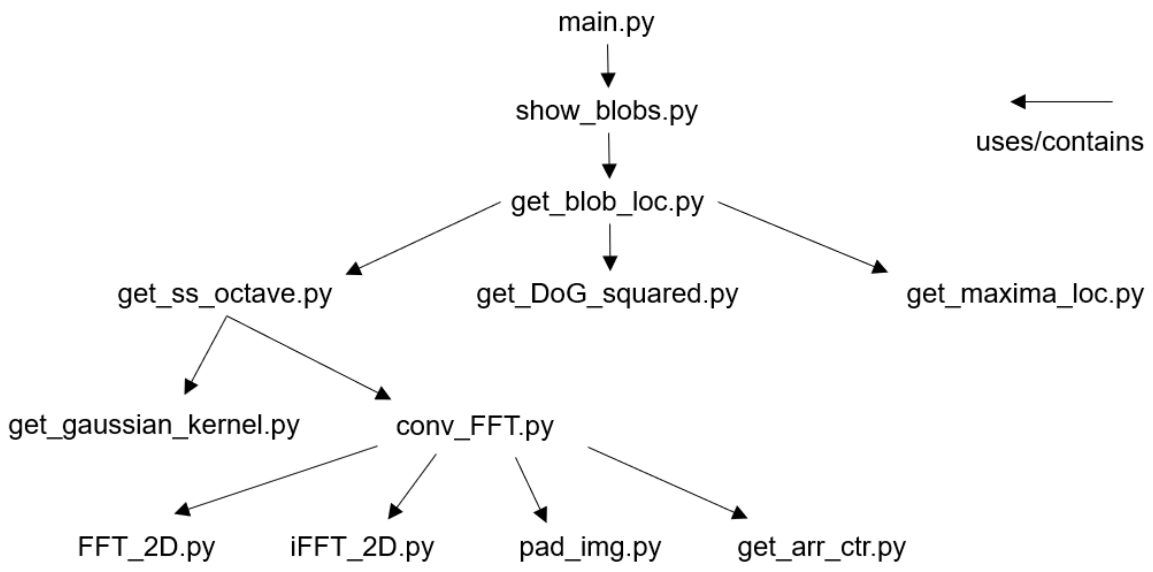

Figure 1.1 – Function Dependencies

2 Implementation

2.1 Step 1

The first octave in scale space is computed by blurring the original image at using different σ values. More precisely, the original image is blurred using $\sigma, k\sigma, k^2\sigma, \ldots, k^{m-1}\sigma$, where m is the number of layers in the octave. The default value of k is $\sqrt{2}$. This is done using the get_ss_octave.py function. Moreover, this function uses the function get_gaussian_kernel.py to compute a spatial-domain gaussian kernel with dimensions that are dependent on σ. Once this kernel is computed, the conv_FFT.py function then filters the original image with different σ values using the Fast Fourier Transform. More precisely, the conv_FFT.py function transforms the input image and kernel to the frequency domain, then performs frequency-domain filtering. Additionally, the conv_FFT.py function is capable of filtering an image and then preparing it for display, like the OpenCV function filter2D(). More details can be found in the comments in the get_ss_octave.py, get_gaussian_kernel.py, and conv_FFT.py files.

2.2 Step 2

Once the first octave in scale-space is computed, the get_DoG_squared.py function is used to compute the squared Difference of Gaussians. This function takes a scale-space octave as an input and outputs the squared Difference of Gaussians. It does by iterating through octave and subtracting every two consecutive layers, then element-wise squaring the result. Figure 2.1 below summarizes this procedure:

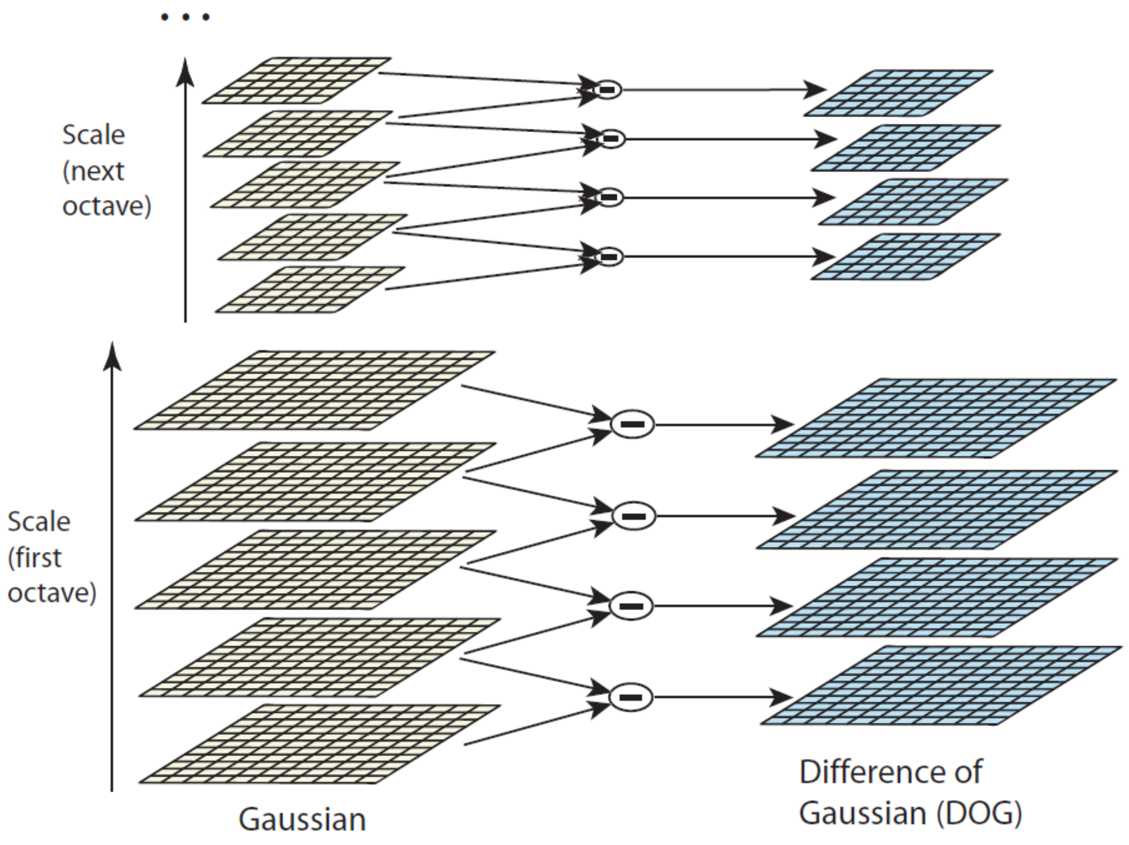

Figure 2.1 – Difference of Gaussians

More details can be found in the comments in the get_DoG_squared.py file.

2.3 Step 3

Once the octave of squared Difference of Gaussians is computed, the maxima in each layer are located using the get_maxima_loc.py function. This function takes in an octave of squared DoG, and outputs the indices of the maxima and the layer number in which they occur. The first 3 layers in the octave are initially stacked together. Next, a global threshold value is computed in the middle layer using Yen's method. Note, however, that in the comments in the get_maxima_loc.py file, different thresholding methods were also tested, such as Otsu's method and the statistical mean. Once this thresholding value is calculated, the 3 layer stack is padded with 1 row before and after and 1 column before and after of zeros. This is done to make room for a 3x3x3 window view into the stack. This 3x3x3 window is shown in Figure 2.2:

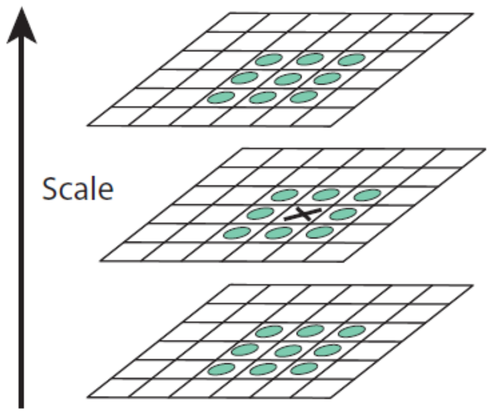

Figure 2.2 – 3x3x3 window view

This window is then slid over each pixel in the stack, and the pixel in the center in the middle layer, marked by the X symbol in Figure 2.2, is compared to the maximum value in the 3x3x3 window. Additionally, it is also compared to the global thresholding value to ensure that it is a maximum. If the pixel marked by the X symbol is equal to the maximum value in the 3x3x3 window, and it is greater than the threshold value, then it recorded as a maximum. More details can be found in the comments in the get_maxima_loc.py file.

2.4 Step 4

Once the maxima in the squared DoG octave have been located, the third image from the bottom of the scale-space octave is down-sampled to obtain the first image in the second octave in scale-space. Since the default value of k is $\sqrt{2}$, then $k^2 = 2$, which means that the third layer in the first scale-space octave is blurred with $k^2\sigma = 2\sigma$. According to [1], this is the first image in the second octave. This down-sampling is done get_blob_loc.py file.

2.5 Step 5

The previous steps are repeated until the desired number of octaves is generated. More details can be found in the comments in the get_blob_loc.py file.

2.6 Step 6

The show_blobs.py function is used to overlay circles around the detected blobs. First, the functions obtains the locations of the maxima using the get_blob_loc.py function, then it uses the OpenCV functions cv.circle() and cv.imshow() to display the overlaid circles. More details can be found in the comments in the show_blobs.py function.

3 Test Images

First, the butterfly.jpg image was tested:

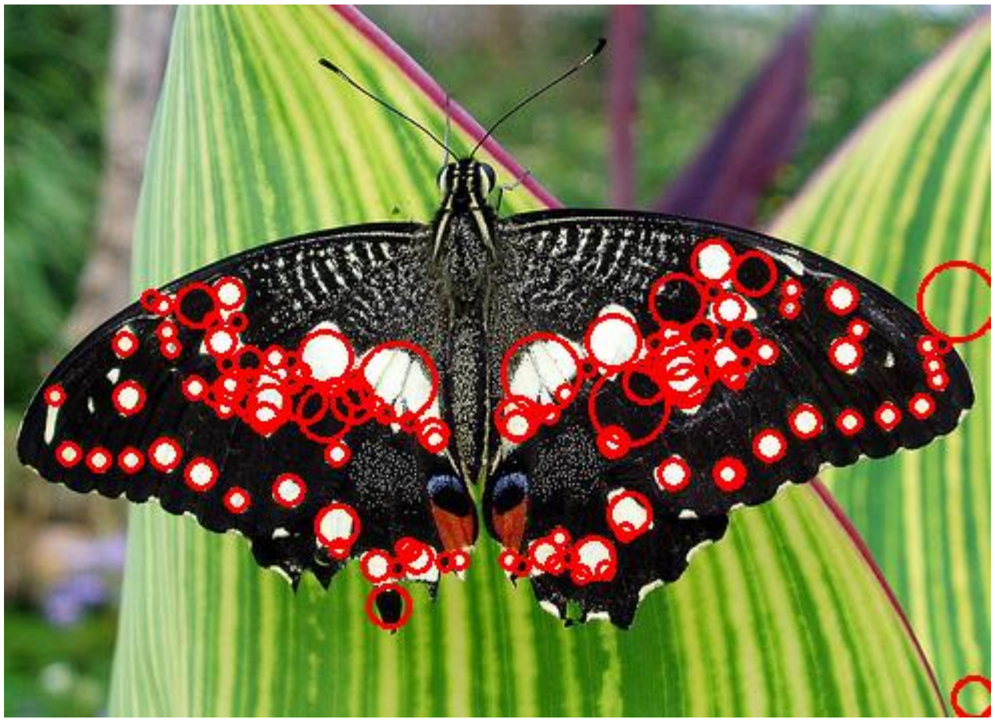

Figure 3.1 – Butterfly.jpg using Yen's thresholding method

Note that Yen's thresholding method was used to generate Figure 3.1. Additionally, Otsu's thresholding method and the mean were used to threshold, as shown in Figures 3.2 and 3.3.

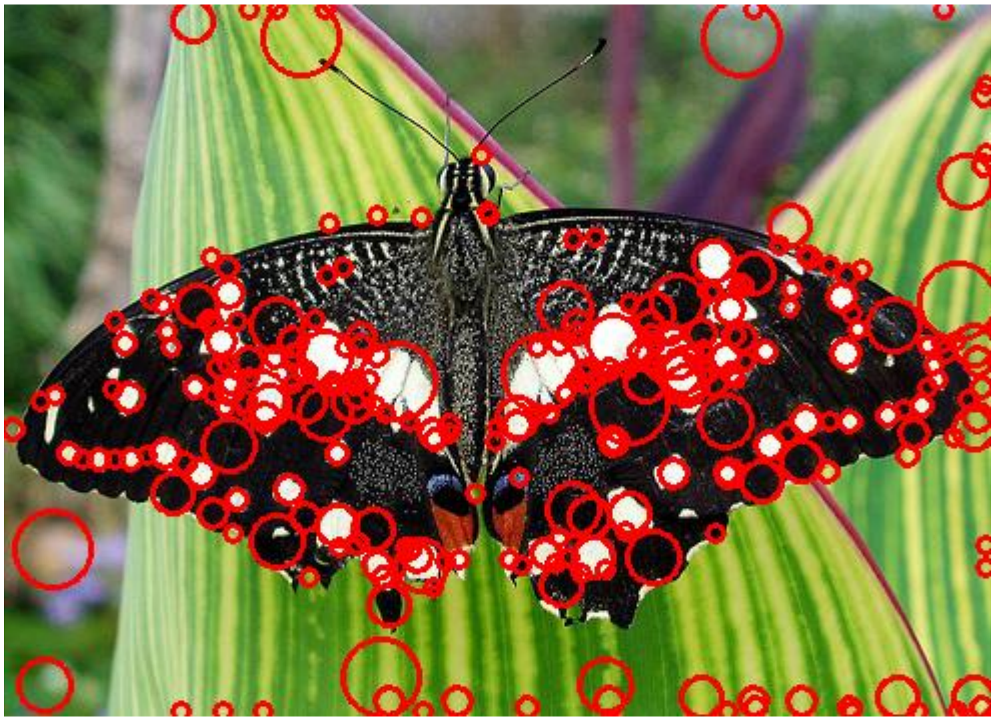

Figure 3.2 – Butterfly.jpg using Otsu's thresholding method

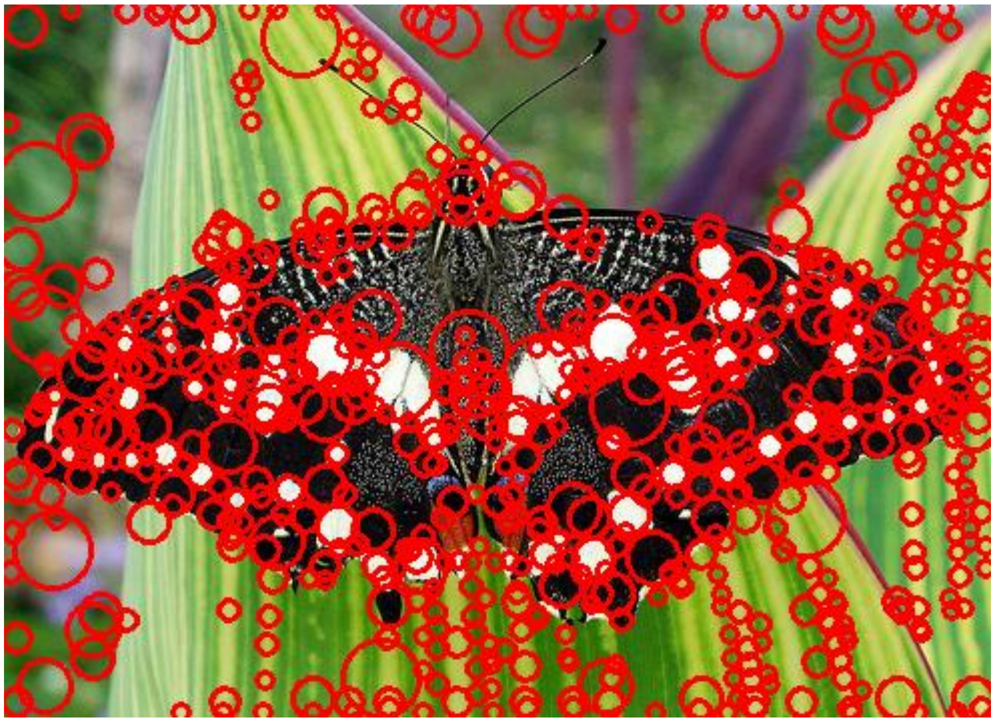

Figure 3.3 – Butterfly.jpg using the mean to threshold

It can be seen from Figures 3.1,3.2, and 3.3 that Yen's method is the most precise in detecting blobs. However, it is susceptible to false negatives, where it misses blobs. Otsu's method contains more false positives than Yen’s method. However, it can detect more blobs. Finally, using the mean to threshold leads to many false positives, so it is not a reliable thresholding method.

Next, the sunflowers.jpg image was tested using Yen’s method:

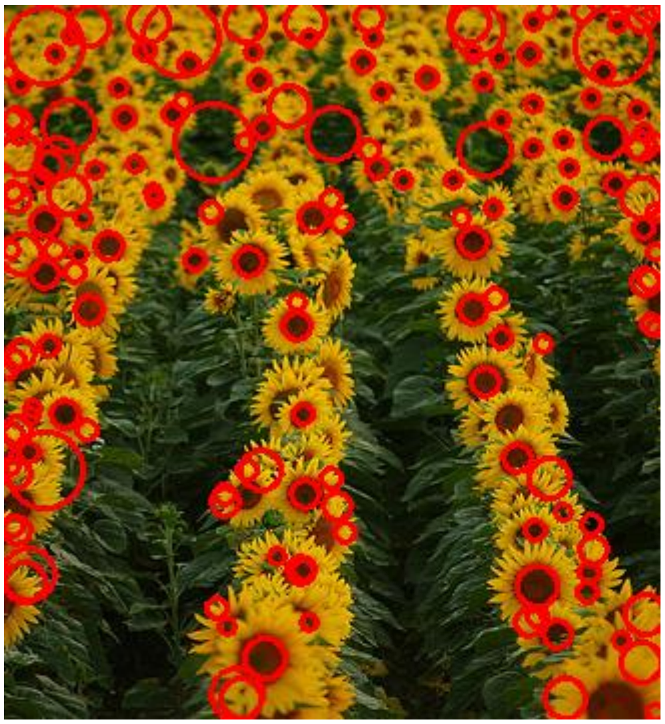

Figure 3.4 – Sunflowers.jpg using Yen’s thresholding method

Next, the fishes.jpg image was tested using Yen’s method:

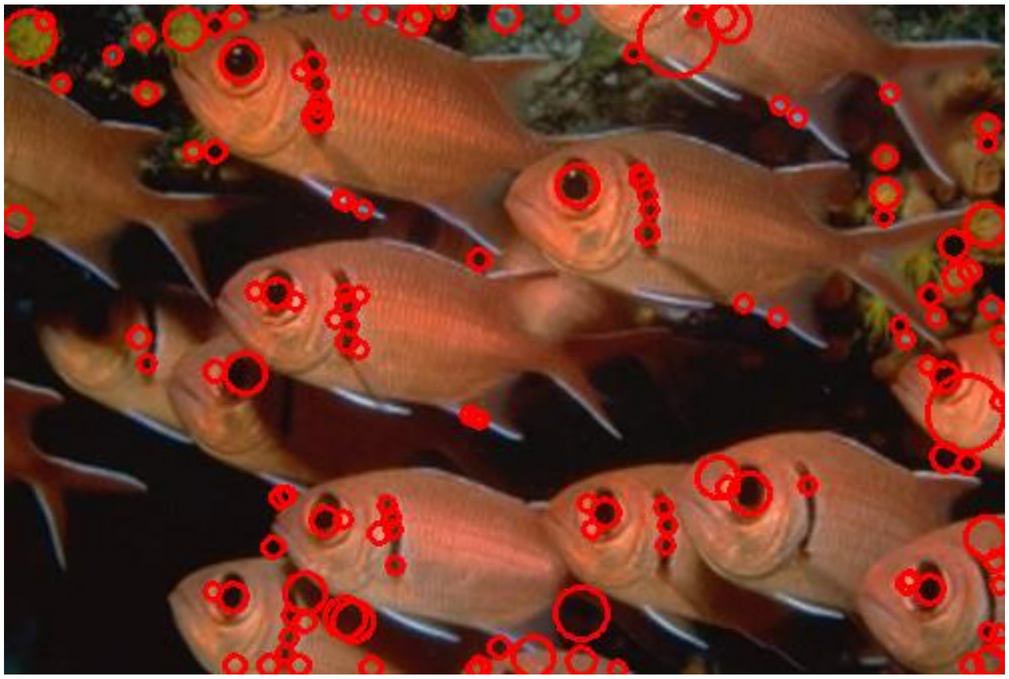

Figure 3.5 – Fishes.jpg using Yen’s thresholding method

Next, the einstein.jpg image was tested using Yen’s method:

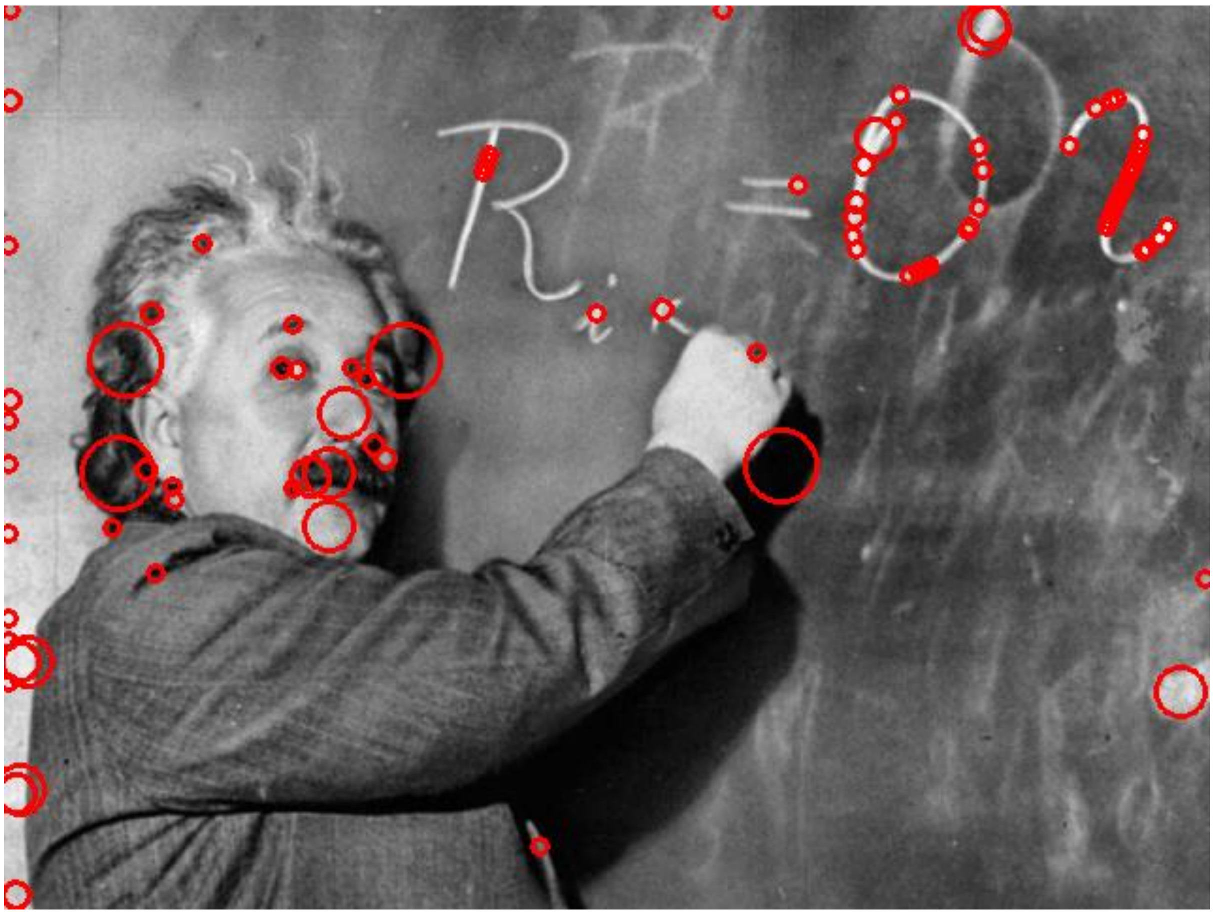

Figure 3.6 – Einstein.jpg using Yen's thresholding method

Next, the following image:

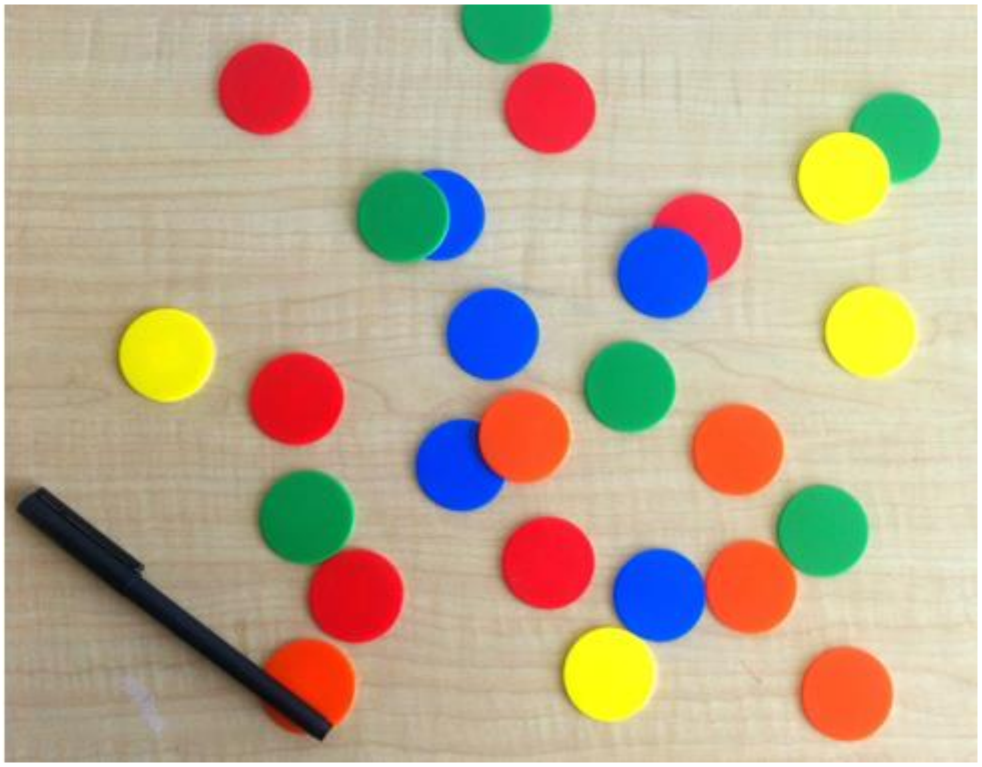

Figure 3.7 – Picture of coins

Was tested using Otsu's method:

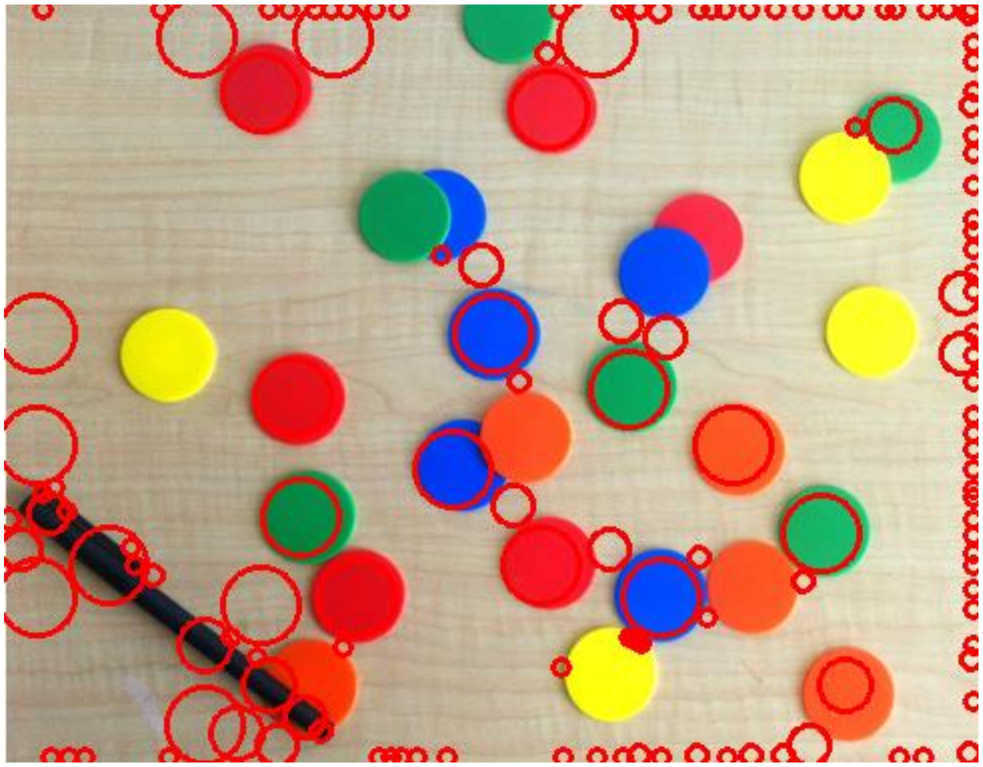

Figure 3.8 – Picture of coins tested using Otsu's method

Next, the following image:

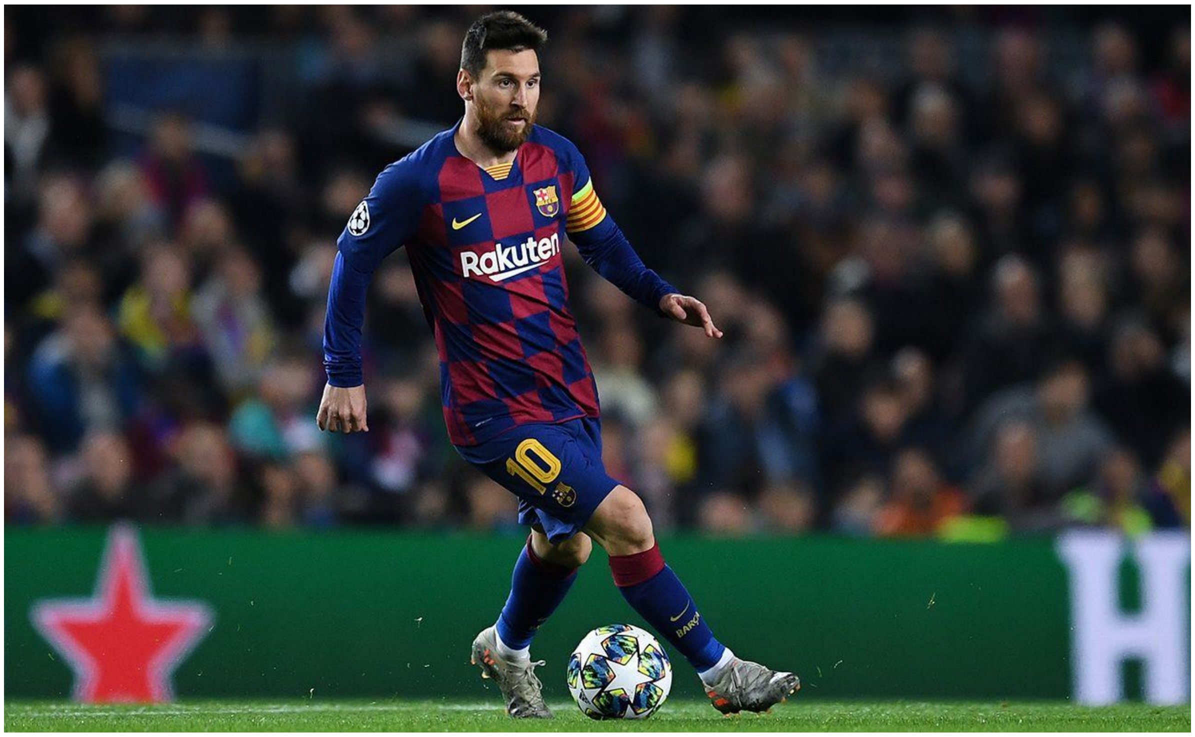

Figure 3.9 – Picture of Lionel Messi

Was tested using Yen's method:

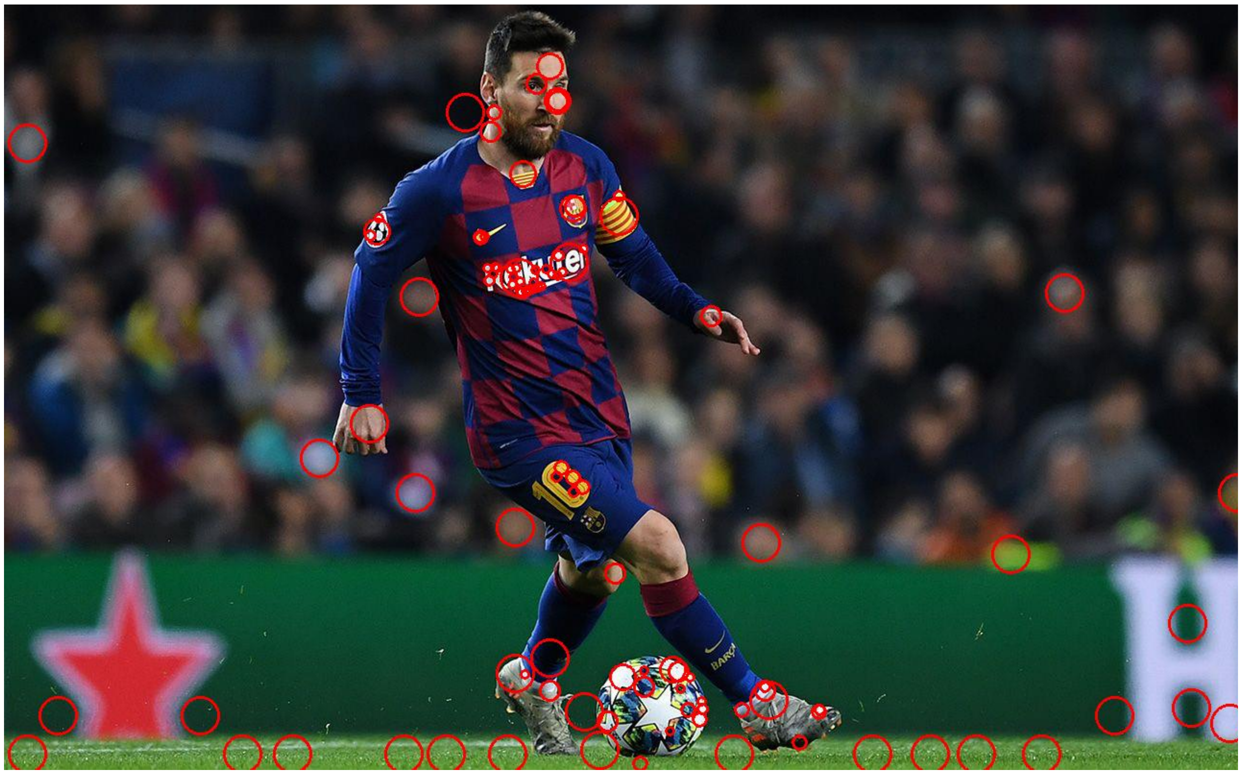

Figure 3.10 – Picture of Lionel Messi tested using Yen's method

Next, the following image:

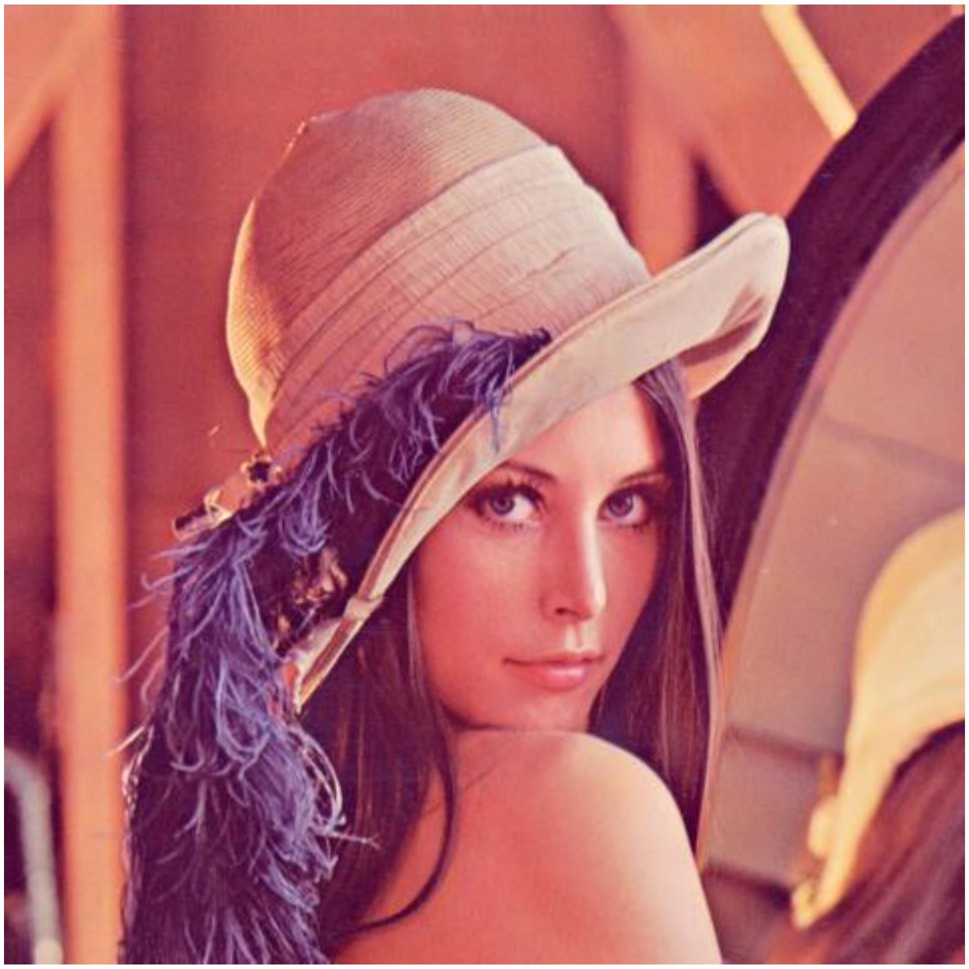

Figure 3.11 – Picture of Lena

Was tested using Otsu’s method:

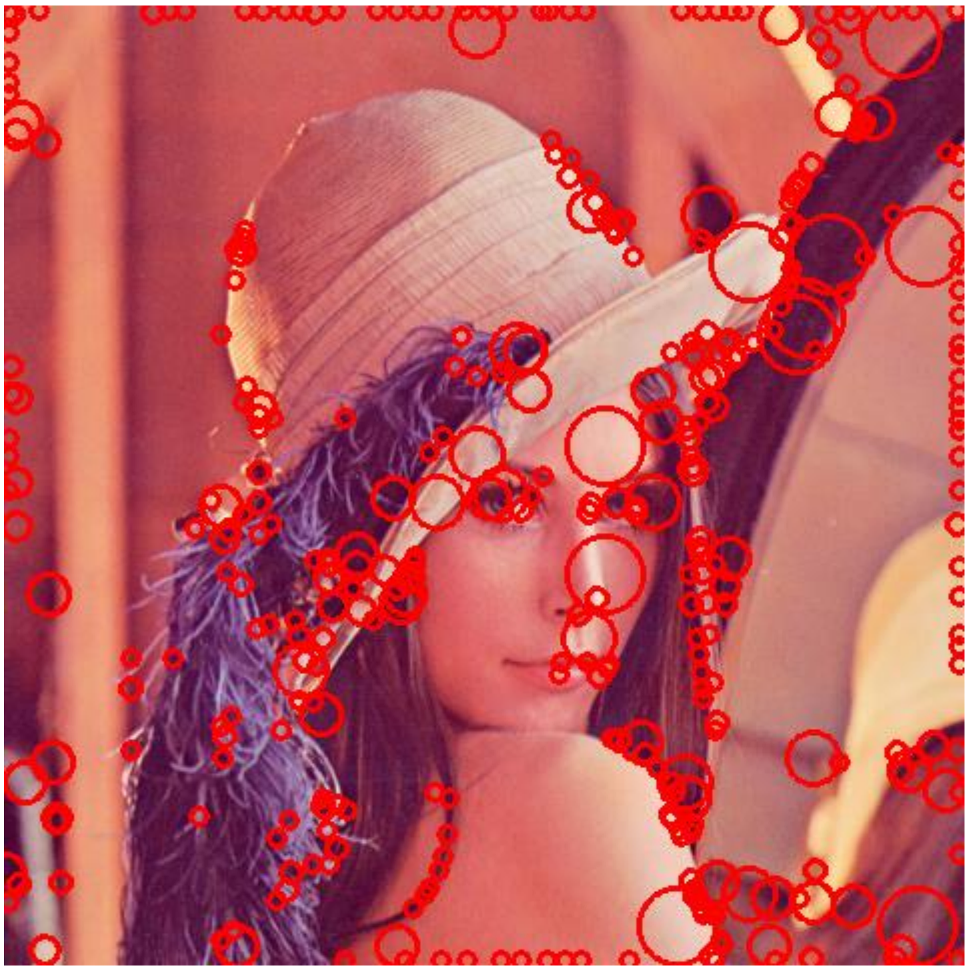

Figure 3.12 – Picture of Lena tested using Otsu's method

Next, the following image:

Figure 3.13 – Picture of Rafael Nadal

Was tested using Yen’s method:

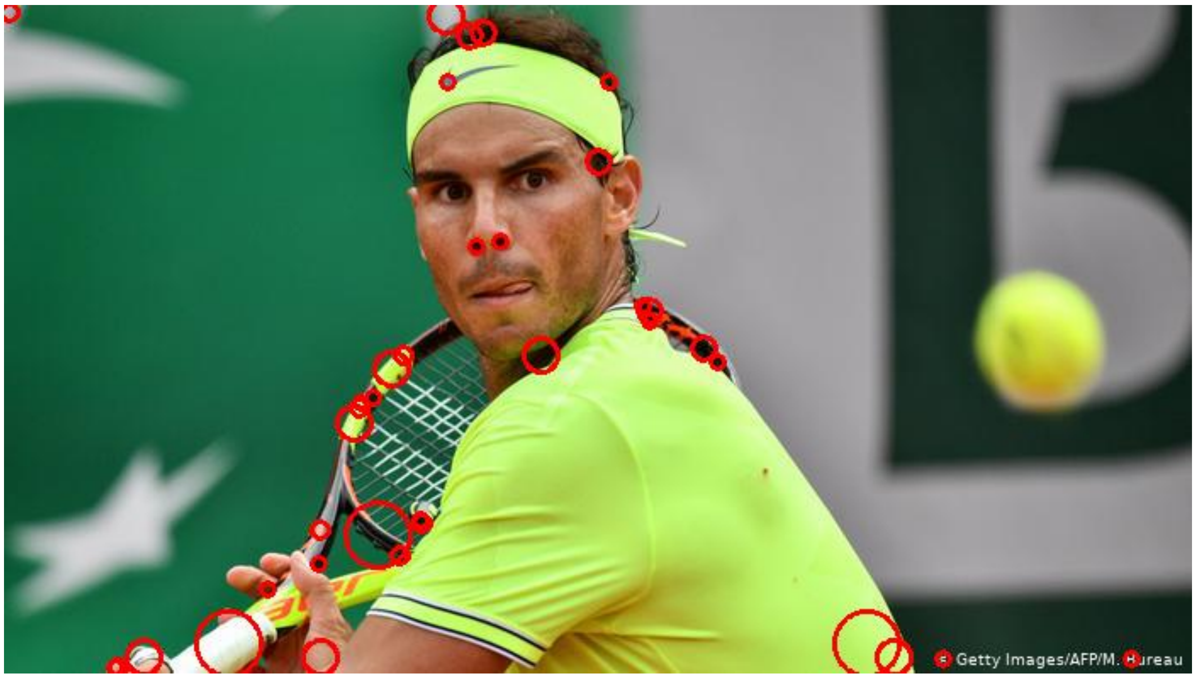

Figure 3.14 – Picture of Rafael Nadal tested using Yen’s method

4 References

[1] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," University of British Columbia, 2004.