Julio Soldevilla
EECS 504 Winter 2018 — Problem Set 2

Problem 1 Problem1

Proof:

1. For an image of the given size $n \times n$ and a Gaussian kernel of size $(2k+1)\times(2k+1)$, when we apply the kernel to just one full window in the image, we have $(2k+1)^2$ multiplications and $(2k+1)^2 - 1$ additions. Then, to go over the whole picture, we need to make the convolution with $(n-(2k+1)+1)^2$ windows, and so this implies that in total the total number of operations to convlve the imgabe by the kernel is $(n-2k)^2 * ((2k+1)^2 + (2k+1)^2 - 1)$.

2. Consider the $2D$ Gaussian kernel $G_{2D}(s,t) = \frac{1}{2\pi\sigma^2} e^{-\frac{s^2+t^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{s^2}{2\sigma^2}} \times \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}} = G_{1D}(s) \times G_{1D}(t)$ where we define $\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{s^2}{2\sigma^2}} = G_{1D}(s)$ and $\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}} = G_{1D}(t)$ are the 1D Gaussian Kernel's. Then, if we convolve $G_{2D}(s,t)$ with some function f, we have that $G_{2D}(s,t) * f = (G_{1D}(s) \times G_{1D}(t)) * f = G_{1D}(s) \times (G_{1D}(t) * f)$ and we see that convolving with 2D Gaussian is the same as convolving with 1D Gaussian twice.

3. Replacing the $2D$ Gaussian kernel with the 1d Gaussian kernel would imply that for applying the 1D horizontal kernel to one window of the corresponding size we would do $(2k+1)$ multiplications and $2k$ additions. Then, we can do this operation in a row for $(n-2k)$ times and then we need to repeat this process n times. So, we end up having to do $n(n-2k)[(2k+1)+2k]$ operations for applying the horizontal 1D Gaussian Kernel. Then, we will have the exact same count of operations for the vertical 1D Gaussian Kernel. Thus, for applying the 1D Gaussian Kernel twice we would have to do $2n(n-2k)[(2k+1)+2k]$ operations.

   In the case of a 2D kernel of 5×5, applying the 2D kernel we would need $(n-4)^2 * 49$ operations to convolve the image with the kernel. Applying the 1D kernel twice we would need $18n(n-4)$ operations. Then, we can see that there will definitely be some n such that applying the $2D$ kernel is cheaper. TO figure out the exact amount, we just set the equations equal and find the value of n. Doing this, we et $n = \frac{196}{31} \approx 6.3$ Thus, when $n \geq 7$, it is cheaper to apply the $1D$ Gaussian Kernel twice instead of the $2D$ Kernel.

4. Now, recall that the LoG operator is given by the mask: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ or $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$. For a Kernel to be separable, we must be able to find two vectors whose outer product is the kernel shown above, which is the same as saying that the rows of the matrix representing the kernel are linearly dependent. We can clearly see that this is not the case in the LoG and so we conclude that the LoG is not separable.

Recall we can approximat ethe LoG with Difference of Gaussians. One thing we can do to ahiceve an almost equivalent operation as if using LoG is to do a difference of 2D Gaussians and use the separability of this operator to get a cheaper operator than using the LoG but still with similar results.

5. Finally, let $\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$. Notice that

$$u'^2 = u^2(\cos^2(\theta)) - 2uv\cos(\theta)\sin(\theta) + v^2\sin^2(\theta) \tag{1}$$

and

$$v'^2 = u^2(\sin^2(\theta)) + 2uv\cos(\theta)\sin(\theta) + v^2\cos^2(\theta) \tag{2}$$

. Then we have that $(G_1 - G_2)(u', v') = G_1(u', v') - G_2(u', v') = \frac{1}{2\pi\sigma_1^2} e^{-\frac{u'^2+v'^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{u'^2+v'^2}{2\sigma_2^2}} \overset{\text{by eq. 1 and 2}}{=} \frac{1}{2\pi\sigma_1^2} e^{-\frac{u^2+v^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{u^2+v^2}{2\sigma_2^2}} = G_1(u, v) - G_2(u, v)$. Thus, this shows that the DoG operator is rotationally invariant, as desired.

■

Problem 2 Problem 2

Proof:

1. The code for harris is the following:

```
function C = harris(dx,dy,Whalfsize)

% function C = harris(dx,dy,Whalfsize)
%
%     EECS 504;
%     Jason Corso
%
%   I is the image (GRAY, DOUBLE)
%   or
%   dx is the horizontal gradient image
%   dy is the vertical gradient image
%
%   If you call it with the Image I, then you need set parameter dy to []
%
%   Whalfsize is the half size of the window.  Wsize = 2*Whalfsize+1
%
%   Corner strength is taken as min(eig) and not the det(T)/trace(T) as
    in
%   the original harris method.  Just for simplicity
%
%   Output
%   C is an image (same size as dx and dy) where every pixel contains the
%   corner strength.
```

```
23
24
25 if (isempty(dy))
26    im = dx;
27    dy = conv2(im,fspecial('sobel'),'same');
28    dx = conv2(im,fspecial('sobel')',’same’);
29 end
30
31
32 %%% YOU NEED TO FILL THE CODE BELLOW
33 % Corner strength is to be taken as min(eig) and not the det(T)/trace(T)
       as in
34 %  the original harris method.
35
36 Wsize = 2*Whalfsize + 1;
37
38 % This is the matrix of zeroes that we ned to fill out.
39
40 sz = size(dx);
41
42 M = sz(1);
43
44 N = sz(2);
45
46 C = zeros(M,N);
47
48 Struct_Matrix = zeros(2,2);
49
50 % Padding NaN to dx and dy
51
52 zeros_row = zeros([N, Whalfsize]);
53 %size(NaN_row);
54 zeros_vector = zeros([M + 2*Whalfsize, Whalfsize]);
55 %size(NaN_vector);
56
57 dx_zeros = [];
58 dx_zeros_2 = [];
59
60 dx_zeros = [zeros_row.’; dx; zeros_row.’];
61 size(dx_zeros);
62 dx_zeros_2 = [zeros_vector, dx_zeros, zeros_vector];
63 size(dx_zeros_2);
64
65 dy_zeros = [];
66 dy_zeros_2 = [];
67
68 dy_zeros = [zeros_row.’; dy; zeros_row.’];
69 dy_zeros_2 = [zeros_vector, dy_zeros, zeros_vector ];
70
71
72 size_NaN = size(dy_zeros_2);
73 M_NaN = size_NaN(1);
```

```
N_NaN = size_NaN(2);


% In the following lines we build the Structure tensor and take the
% eigenvalue of the matrix.

for i = (Whalfsize+1): M - Whalfsize
    for j = (Whalfsize + 1): N - Whalfsize
        for k = 1:3
            if k == 1
                I_xsum = 0;
                for p = -Whalfsize:Whalfsize
                    for q = -Whalfsize:Whalfsize
                        I_xsum = I_xsum + dx(i+p,j+q)*dx(i+p,j+q);
                    end
                end
                Struct_Matrix(1,1) = I_xsum;
            elseif k == 2
                I_xysum = 0;
                for p = -Whalfsize:Whalfsize
                    for q = -Whalfsize:Whalfsize
                        I_xysum = I_xysum + dx(i+p,j+q)*dy(i+p,j+q);
                    end
                end
                Struct_Matrix(1,2) = I_xysum;
                Struct_Matrix(2,1) = I_xysum;
            elseif k == 3
                I_ysum = 0;
                for p = -Whalfsize:Whalfsize
                    for q = -Whalfsize:Whalfsize
                        I_ysum = I_ysum + dy(i+p,j+q)*dy(i+p,j+q);
                    end
                end
                Struct_Matrix(2,2) = I_ysum;
            end
        end
        %Struct_Matrix;
        minimum_eigenvalue = min(eigs(Struct_Matrix));
        C(i,j) = minimum_eigenvalue;
    end
end

C;
end


%%% YOU NEED TO STOP HERE
```

The output images are:

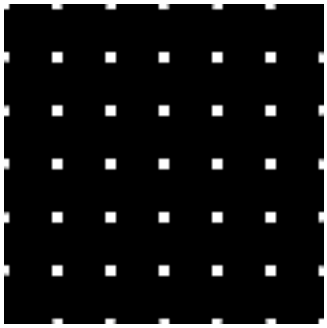

Figure 1:

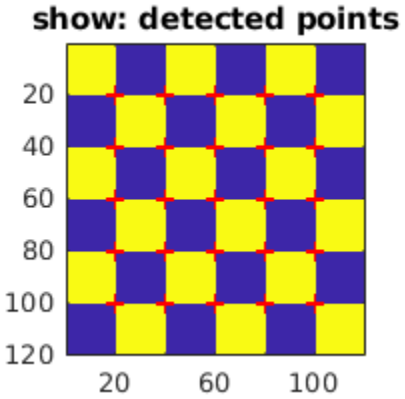

Figure 2:

2. Running run_3_2.m generates the following image:

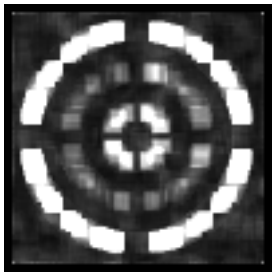

Figure 3:

This image looks like this because the original image is not completely smooth, so the Harris detector is actually detecting this non smoothness of the figure. The Harris detector is considering the brusque changes in the intensity of the pixels forming the image as corners of

the circles and so that is why there are so many responses in the picture. Since the image is not smooth (in terms of changes of intensities in the pixels) the detector has a response for places that are not in the circles. No full ring has a high corner response over the entire ring, because when we are in the north, south, east and west parts of the circles, the pixels of very similar intensity are exactly stacked one over the other or one next to the other and so for this small regions of the circles there are no brusque changes in the values of the pixels and so in these regions, the operator doesn't detect these regions as corners and the output doesn't show a high corner response in this part.

3. The code we use for this function is

```
% EECS 504 HW2p2
% run_3_3.m

% rings part b
% this code will call harris, it does post-processing to extract corners
% from the corner response image
X = detect(im,3,'harris',1,0.1,5,true);
show(im,X)
fig = gcf;
fig.PaperUnits = 'points';
fig.PaperPosition = [0 0 150 150];
print('detect_rings.png','-dpng','-r0');
```

The image this changed code generates is:

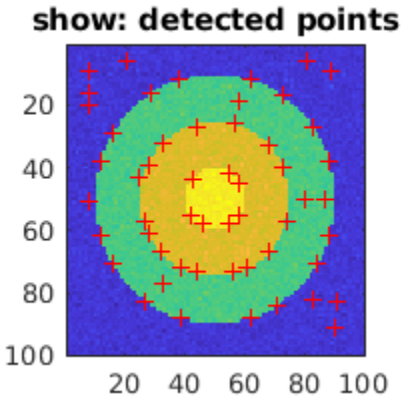

Figure 4:

In this code, we can see that the Harris operator detected corners only on the boundaries. This happens because in the detect function in the write up, change false to true makes the program convolve the image with a Gaussian smoothing kernel that will smooth the corner-like features that were detected before by the Harris operator.

4. The code script run_red_varyk.m returns the following six images:

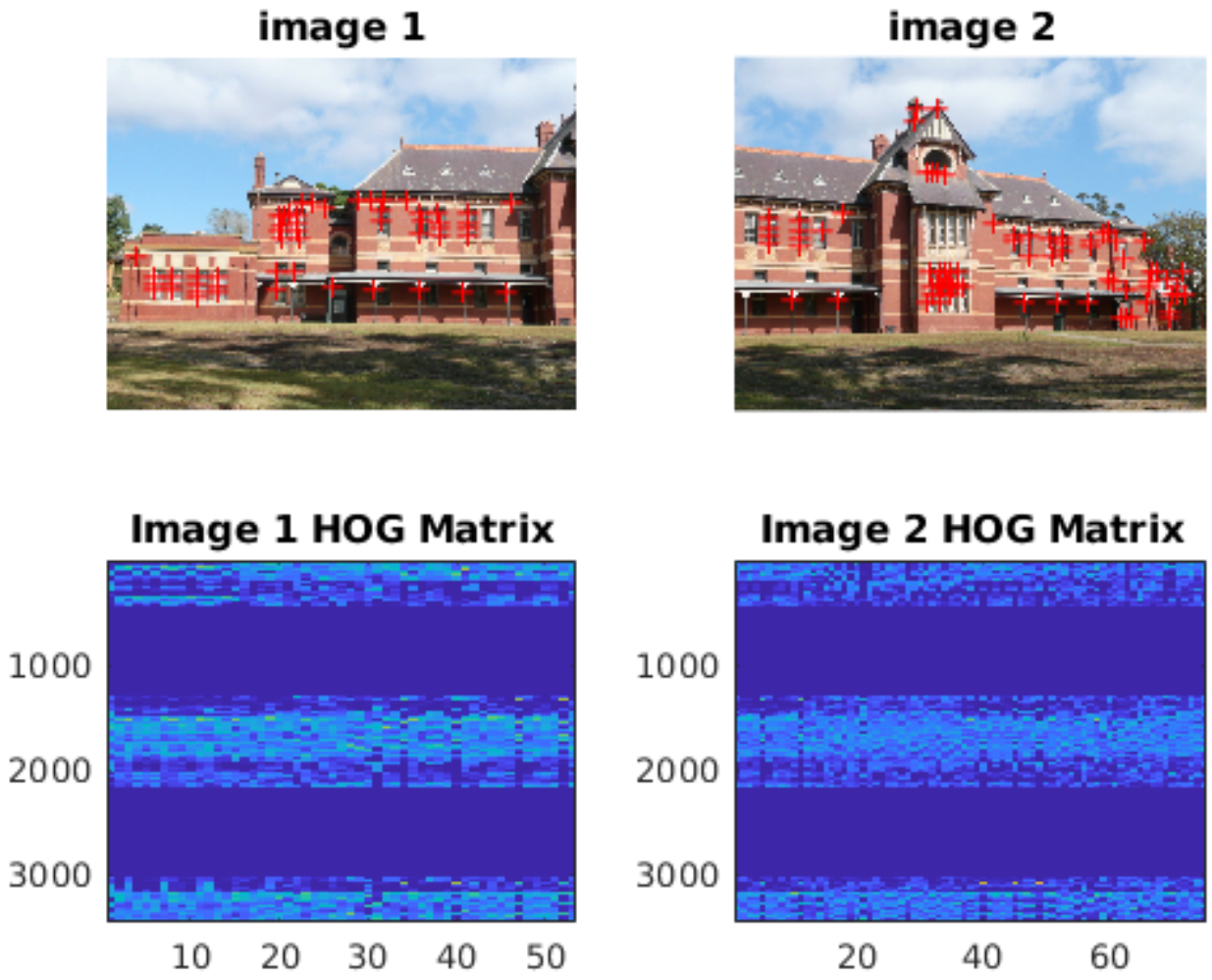

Figure 5: Red correspondences with 4 points

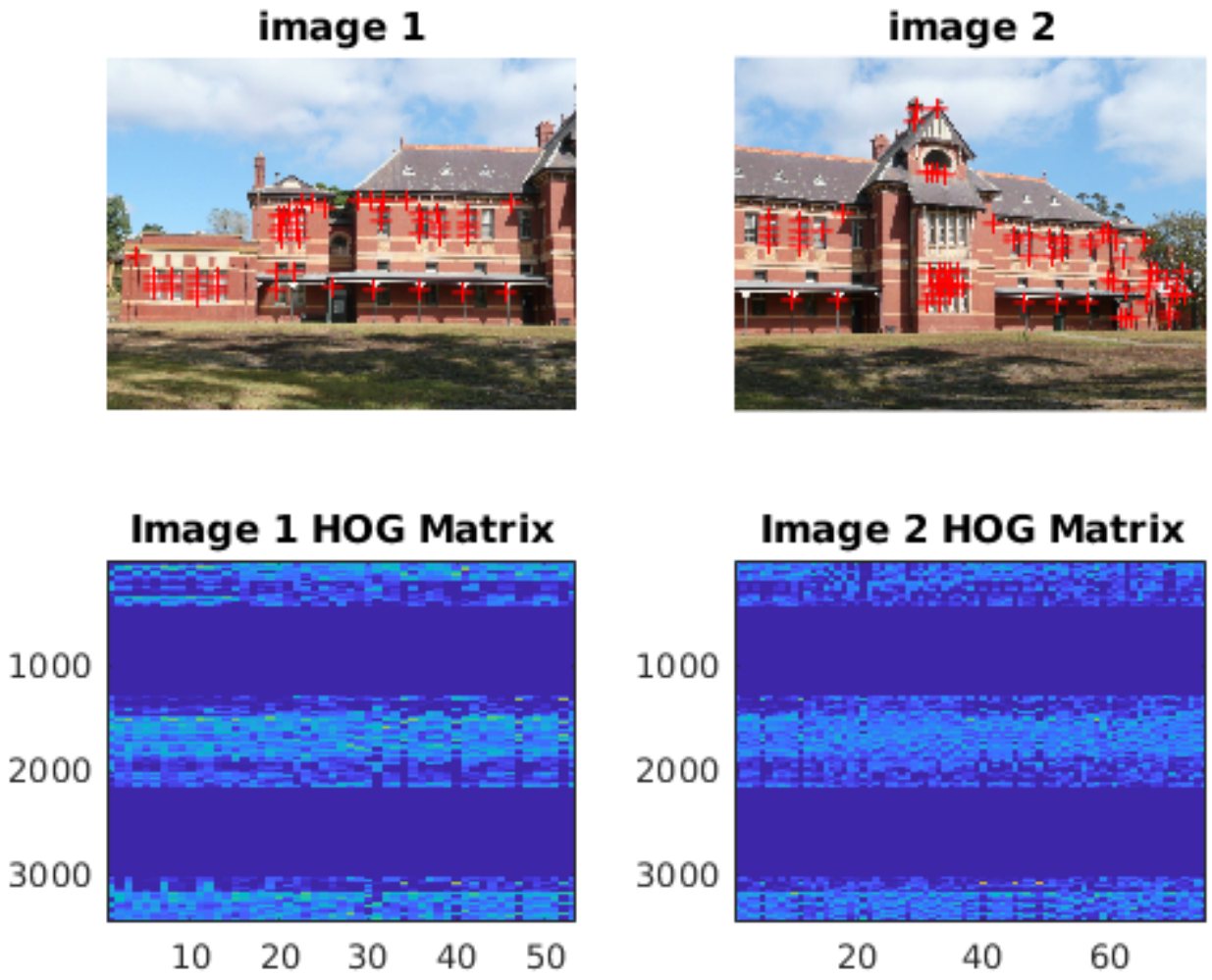

Figure 6: Red correspondences with 10 points

Figure 7: Red Correspondences with 20 points.

Figure 8: Red stichted with 4 points.

Figure 9: Red stichted with 10 points.

Figure 10:

IN this case, we can see that the best correspondence comes when we use 10 points. This happens because with 10 points there is information to match the picture correctly and therefore the estimation is optimal. When we have only 4 points, we don't have enough points to match the pictures adequately and so the estimation is not accurate, that is why we end up having the picture shown above. Finally, with 20 points, we now have too much information, namely too many points, and so the code cannot process so much information and ends up mismatching the points and therefore there is a wrong estimation of the homography that aligns the image.

■

Problem 3 *Problem 3*

Proof:

1. The DoG Scale space for the circle is the following:

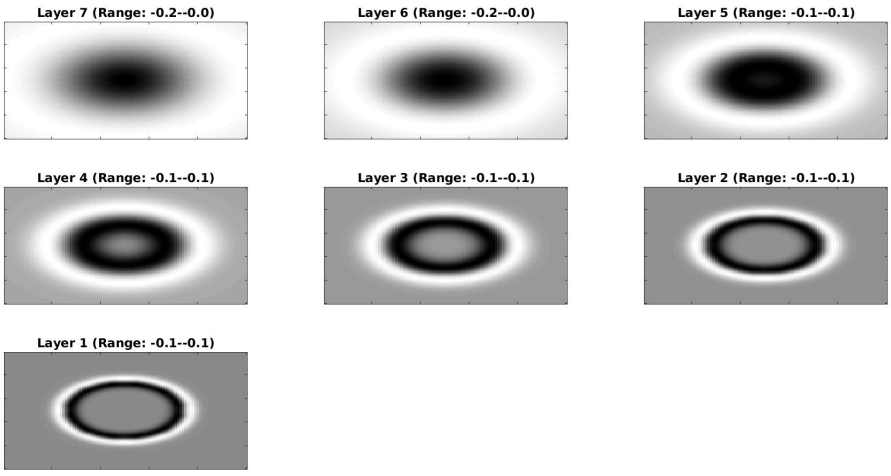

Figure 11:

The scale space for the sunflower picture is:

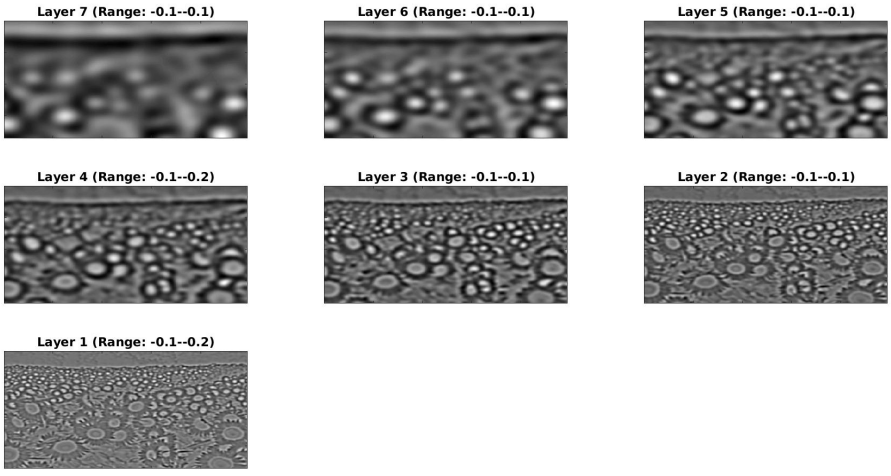

Figure 12:

THe fly brain scale space picture is:

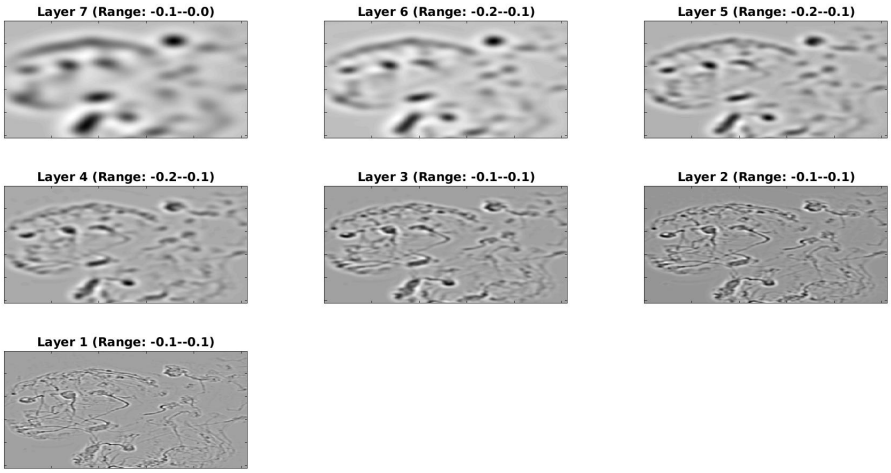

Figure 13:

In the sunflower scale space we can see how the how the centers of the sunflowers are always recognizable, even after blurring in the 7th layer of the DoG scale. A similar thing happens with the circle scale space, which is just a much simpler space than the sunflower space. In the circle space, it is interesting to see how the outer and inner part of the circle are almost of the same color in layer 1, but as we increase in the layers, the center of the circle turns white, the edge becomes whiter and bigger and we see almost nothing of the outside of the edge by layer 7.

2. In the following pictures, we have the visualization of the detected extrema.

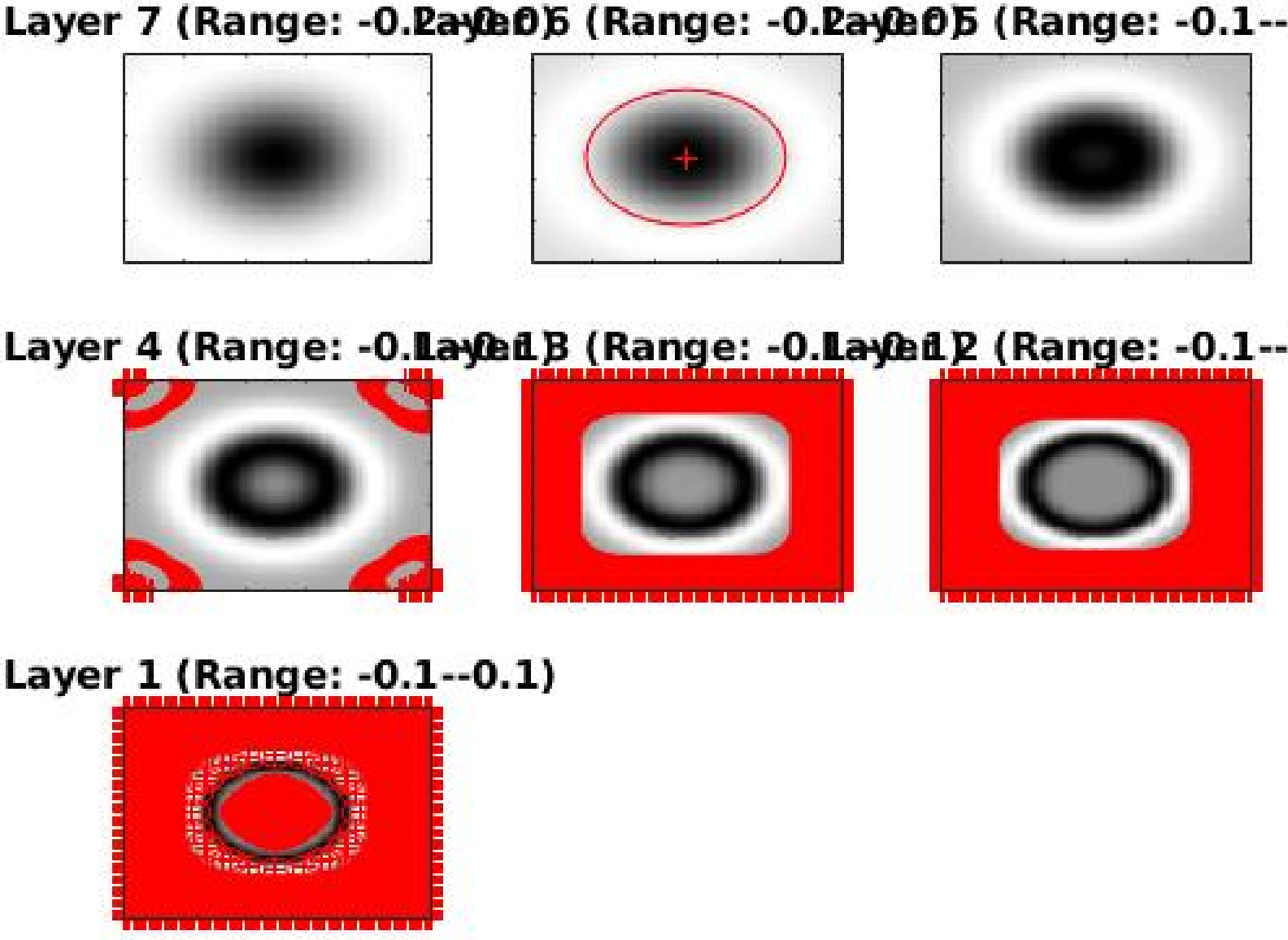

Figure 14:

the visualization of fly maxima is:

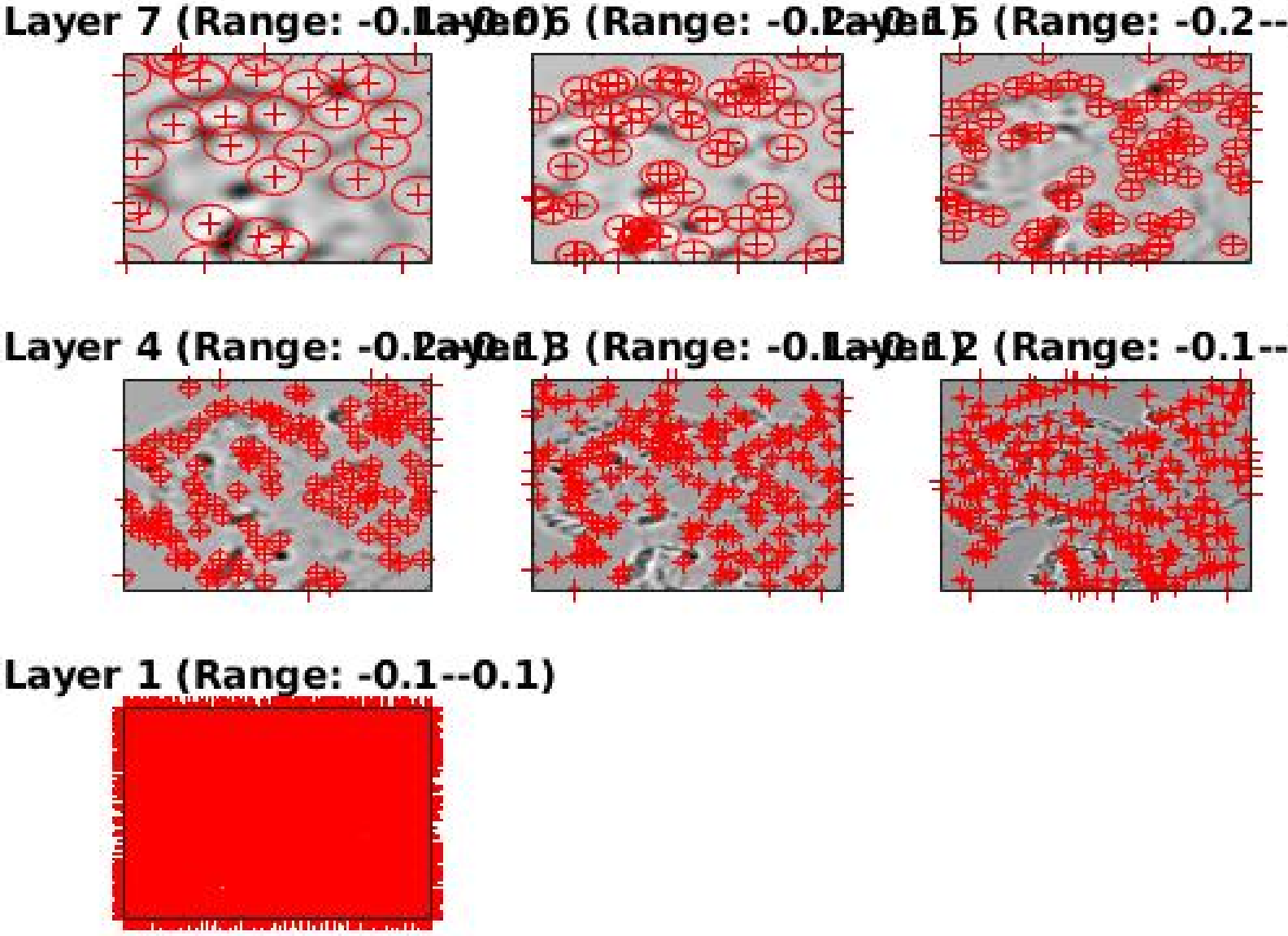

Figure 15:

The visualization of sunflower maxima:

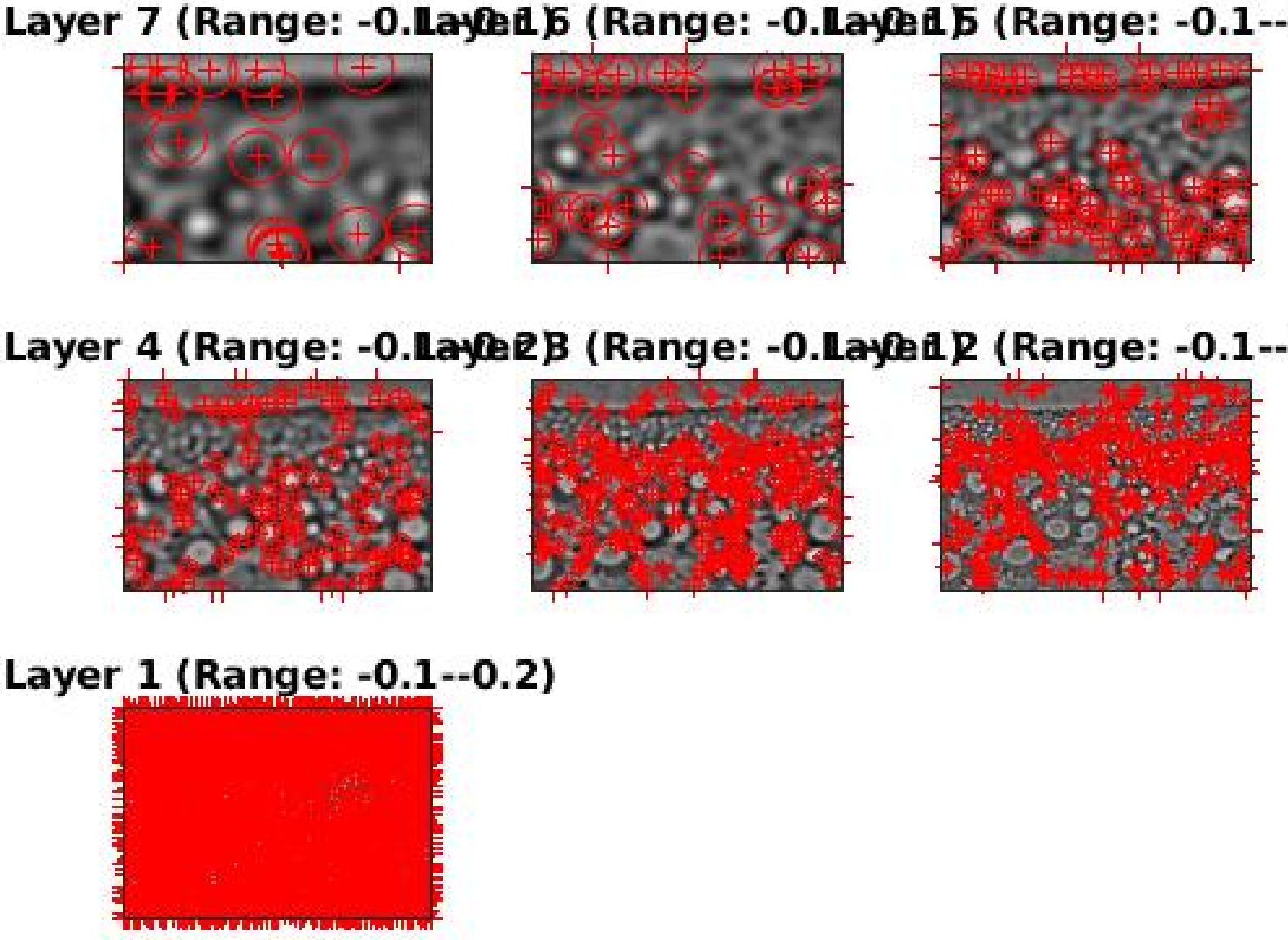

Figure 16:

So in the pictures above, we detect many extrema in the first layers, because in these layers, the σ of the Gaussian is small and so, thinking about the picture of the Gaussian, this is very narrow. Thus, there is a lot of variation in neighboring pixels and this leads us to detect in regions with even small variations a lot of points as maxima and minima points. As the σ

increases, the Gaussian is wider and so there is less brusque variation in neighboring pixels, which implies we will have fewer and fewer maxima and minima. As a ressult, we will detect fewer and fewer maxima and minima in larger regions. That is why we detect less blobs and detect them over wider regions.

3. As a filtering idea, after doing the threshold filtering, I wanted to take the points in the picture corresponding to the remaining detected blobs and take them as center of windows (of size similar to the radius of the blob) and use these windows to compute the Structure tensor to compute the minimum eigenvalue of such a tensor. In this way, I would be able to detect whether there were any corners in the blob centered at the point we picked. If there were corners, we can discard that blob because we want blobs, not corners. And so going through all the points represented in the vector of maxima found in part 2 of the problem, we would filter even more the blobs.

   The code is the following:

```
function E_ = filterBlobs(im,L,E,sa,DoGtau)
%
%  W18 EECS 504 HW2p3 Blob detection
%
% the original scale space blob detector returns all local extrema in
    scale
% space, which tends to return many pixels that do not seem to be blobs.
%
% This function filters out extrema that had too weak a response in the
%  DoG as well as non-blob-like regions.
%
% im is the grayscale, double image in the 0:1 range
% L is the DoG scale space
% E is the Extrema N x 3 matrix  (each row is X,Y,level)
% sa is the sigma vector (roughly 3*sigma is a size in image)
% DoGtau (optional) is the filter on the extrema response

if nargin<5
    DoGtau = 0.1;
end

% number of extrema inputted
n = size(E,1);

[r,c,b] = size(im);

if (b ~= 1)
    fprintf('please supply a grayscale image, double, in range 0:1\n');
    E_ = [];
    return
end

E_filter = [];

%%%% YOU NEED TO FILL IN THE CODE BELOW
```

```
%%%% BE SURE TO FILTER OUT BOTH ON THE RESPONSE OF THE DOG AND THE
%%%% LOCAL IMAGE REGION BLOB-NESS

for i = 1:n
    vector = E(i,:);
    Pos_x = vector(1);
    Pos_y = vector(2);
    Pos_z = vector(3);
    %i
    %[Pos_y,Pos_x,Pos_z]
    %L(Pos_y,Pos_x,Pos_z)
    if abs(L(Pos_y,Pos_x,Pos_z)) > DoGtau
        E_filter = [E_filter;[Pos_x,Pos_y,Pos_z]];
    end
end


%%% Now we try to apply Harris operator and filter even more the maximum
%%% values we found

sz_max = size(E_filter);
sz_max_vector = sz_max(1);

bw_im = im;
size_im = size(im);
M = size_im(1);
N = size_im(2);

E_ = [];

for i = 1:sz_max_vector
    vector = E_filter(i,:);
    point_x = vector(2);
    point_y = vector(1);
    layer = vector(3);
    radius_layer = floor(2.5*sa(layer));

    zeros_row = zeros([N,radius_layer]);
    zeros_vector = zeros([M + 2*radius_layer,radius_layer]);

    im_zeros = [];
    im_zeros_2 = [];
    im_zeros = [zeros_row.'; im; zeros_row.'];
    im_zeros_2 = [zeros_vector, im_zeros, zeros_vector];

    window = im_zeros_2(point_x + radius_layer - radius_layer:point_x +
    radius_layer +radius_layer,point_y+radius_layer - radius_layer:point_y
    + radius_layer +radius_layer);
    window_dx = conv2(window,fspecial('sobel'),'same');
    window_dy = conv2(window,fspecial('sobel')','same');
    Whalfsize = radius_layer;
%     Struct_Matrix = [];
```

```
    for k = 1:3
            if k == 1
                I_xsum = 0;
                for p = -Whalfsize:Whalfsize
                    for q = -Whalfsize:Whalfsize
                        I_xsum = I_xsum + window_dx(Whalfsize +1 + p,
    Whalfsize +1 +q)*window_dx(Whalfsize +1+p,Whalfsize + 1 +q);
                    end
                end
                Struct_Matrix(1,1) = I_xsum;
            elseif k == 2
                I_xysum = 0;
                for p = -Whalfsize:Whalfsize
                    for q = -Whalfsize:Whalfsize
                        I_xysum = I_xysum + window_dx(Whalfsize + 1 +p,
    Whalfsize + 1 +q)*window_dy(Whalfsize + 1 + p,Whalfsize + 1 +q);
                    end
                end
                Struct_Matrix(1,2) = I_xysum;
                Struct_Matrix(2,1) = I_xysum;
            elseif k == 3
                I_ysum = 0;
                for p = -Whalfsize:Whalfsize
                    for q = -Whalfsize:Whalfsize
                        I_ysum = I_ysum + window_dy(Whalfsize + 1 + p,
    Whalfsize + 1 +q)*window_dy(Whalfsize + 1 +p,Whalfsize + 1 +q);
                    end
                end
                Struct_Matrix(2,2) = I_ysum;
            end
        end
        %Struct_Matrix;
        %i
        eigenvalues = eig(Struct_Matrix);
        if min(eigenvalues) > 80
            E_ = [E_; [point_y,point_x,layer]];
        end
end



%%%% YOU NEED TO STOP HERE
```

The picture of the sunflowers after filtering blobs is :

Figure 17:

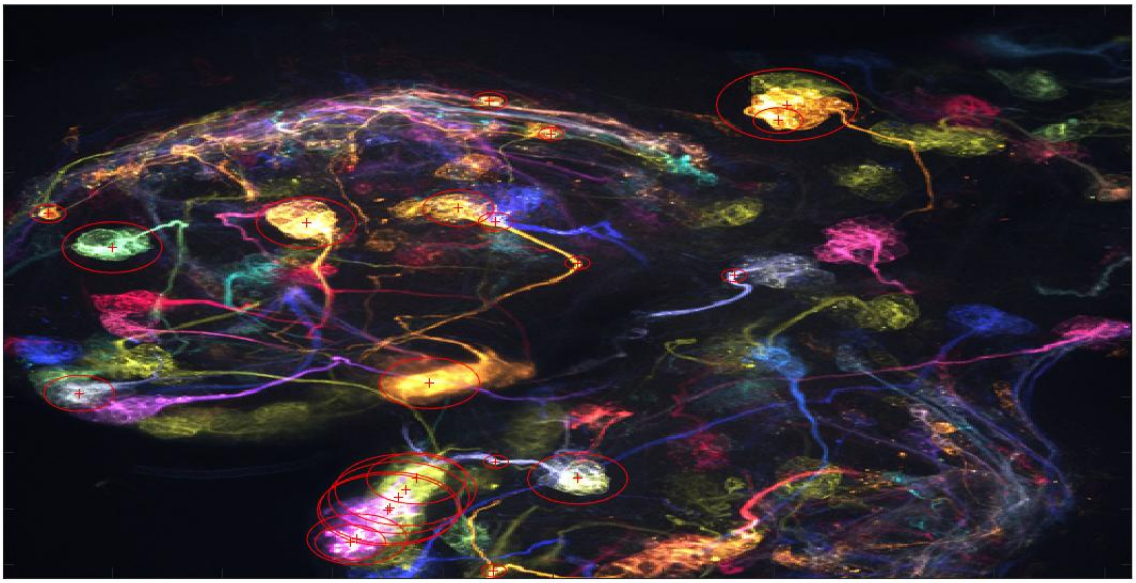

Figure 18:

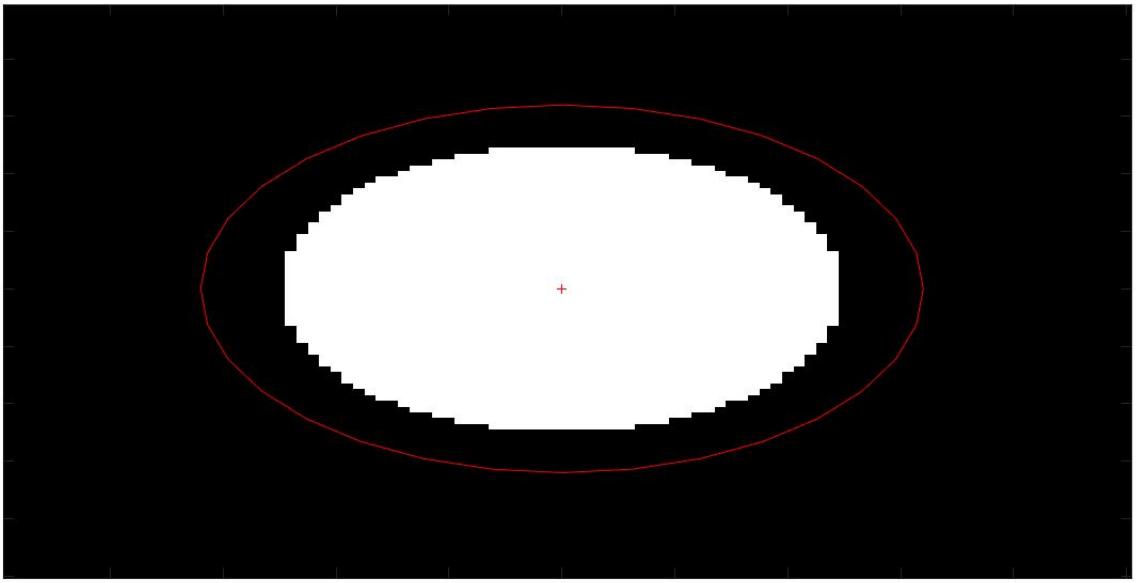

Figure 19: