ECE-558 PROJECT-03

OPTION-3: LAPLACIAN BLOB DETECTOR

Group Members:

1. Aman Waoo (awaoo, 200483401)
2. Pranav Vishnu Anandakumar (pananda, 200482947)
3. Devadharshini Ayyappan (dayyapp, 200482960)

INTRODUCTION:

Blob Detection in image processing is primarily for detecting regions in an image that differs in properties such as color or brightness, compared to surrounding regions. The regions of an image with constant properties are called blobs. This method helps us in finding the complementary information about regions not detected via edge or corner detectors. These regions could signal the presence of objects or parts of objects in the images with application to object recognition or object tracking. Peak detection in segmentation applications can also be achieved with this technique.

ALGORITHM:

1. Creation of Laplacian of Gaussian filter
2. Building a Laplacian scale space and performing convolution
3. Performing Non-Max Suppression in scale space
4. Displaying the resulting circles at their characteristic scales

IMPLEMENTATION OF ALGORITHM:

1. Creation of Laplacian of Gaussian filter:

The Laplacian of Gaussian (LoG) filter is used to highlight regions of rapid intensity changes in an image. Typically, convolution of the Laplacian and the Gaussian filter gives the LoG filter. And it can be mathematically written as:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

In this project, we have used the fspecial() function to create the LoG kernel which takes the input as the filter size and standard deviation. To normalize the LoG, the output function is multiplied by the variance.

Here, the filter size varies based on the sigma value as below:

Filter size = Max(1,6*sigma)

We chose the sigma value as 1.3 after experimenting with a range of values from 1 to 3. For higher sigma values, more redundancies and higher runtime was observed while for the smaller values although the runtime was less, some features were missed. For sigma with a value of 1.3, the program was fast enough and with clear detections without many redundant blobs.

2. Building a Laplacian Scale Space and Convolution

The LoG response is not tested for maxima at the initial and final set value. So, the scale space size is incremented by two and we are changing the size of the input image with a fixed kernel size. After downscaling the input image, we perform the convolution with the LoG filter and square the resultant image. Then, resize the convolved image to the size of the original image and store it in a 3D matrix, space_scale, during each iteration.

3. Performing Non-Max Suppression in scale space

The Non-Max Suppression for the resultant image from the above step is implemented based on the method suggested by Harris [3]. After a number of trials, we have taken the threshold value of 1000, as it typically lies around 1000 for Harris combined corner and edge detector, and seems to be working appropriately for our input images. This is done for all the resultant images and the co-ordinates of the maxima is stored in a separate cell matrix. Now, for each response stored in the 3D matrix, we use the non-max suppression to remove the redundancy of the repeating co-ordinates. Note that the maximas detected at the edges of the image are not used as blob features. The radius of the blob at every maxima co-ordinates is saved and it is relative to the overall size of the image.

4. Displaying the resulting circles at their characteristic scales

Based on the saved maxima co-ordinates and the radius of the blob, we plot the circles on the input image using the viscircles function which exemplifies Blob Detection.

INDIVIDUAL CONTRIBUTIONS:

1. Aman Waoo - Aman worked on generating the Laplacian of Gaussian (LoG) function code, designing the non-max suppression function for evaluating maximas in 2D slices separately, and the main function for fetching image input from the user. He also contributed in preparing the report for algorithm implementation and formatting the code.

2. Pranav Vishnu Anandakumar - Pranav worked on optimizing the code for implementing convolution operation and on designing the non-max suppression function for storing the maximas in 3D scale space matrix. He also contributed in preparing the report by including the references section and formatting the code. Apart from that, he also worked on viscircles() function for creating the circles on the image.

3. Devadharshini Ayyappan - Devadharshini worked on developing the code for rescaling the images according to the kernel size. She also tested the code for various input images and finalized the parameters for in_val, fin_val, scale_steps and std_dev. She also helped in preparing the report by adding the introduction section and formatting the code.

CODE IMPLEMENTATION:

MATLAB Version: R2022a

Script Development: MATLAB Live Editor

Extra Packages Required: None

In order to run the code, simply open the 'awaoo_code.mlx' Live Editor file inside ‘awaoo_code’ folder and click on 'Run all sections'. Select the desired input image from the dialog box and after scrolling down in the script, the output shall be displayed.

```
clear;
clc;

% Get user inputs for image file
[file_full, user_canceled] = imgetfile('MultiSelect', false);

% Display message for no user input
if user_canceled == 1
   disp('User did not select any image.')
else
    [file_path, file_name, file_ext] = fileparts(file_full);
    disp(['Selected image: ', file_full])
end

% Read the image file, convert it to gray scale and extract size of image
img_org = imread(file_full);
img_gray = double(rgb2gray(img_org));
[rows, cols] = size(img_gray);

% Defining the number of scale space steps (by trial and error for
% performance and time trade-offs, steps = 10) and fixing the standard
% deviation to be 1.3 (typical value from 1 to 3)
scale_steps = 10;
std_dev = 1.3;

% Building Laplacian scale space for some initial and final scale values
% The initial and final values shall determine the scaling factor between
% kernel size and original image (typically in_val = 2 and by trial and
% error fin_val = 17)
in_value = 2;
fin_value = 17;

% Start the clock for runtime of the program
tic

% Call blobdetect function
[x_coord, y_coord, radius] = detect_blob(img_gray, std_dev, in_value, fin_value, scale_steps);

% Store the detected blob coordinates to an array and display the original
% image for creating circles on it
blob_xy = [];
imshow(img_org)
hold on
blob_xy(:,2) = x_coord;
blob_xy(:,1) = y_coord;
```

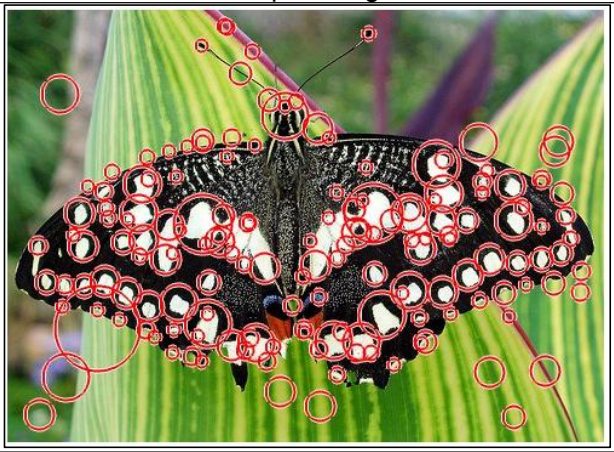
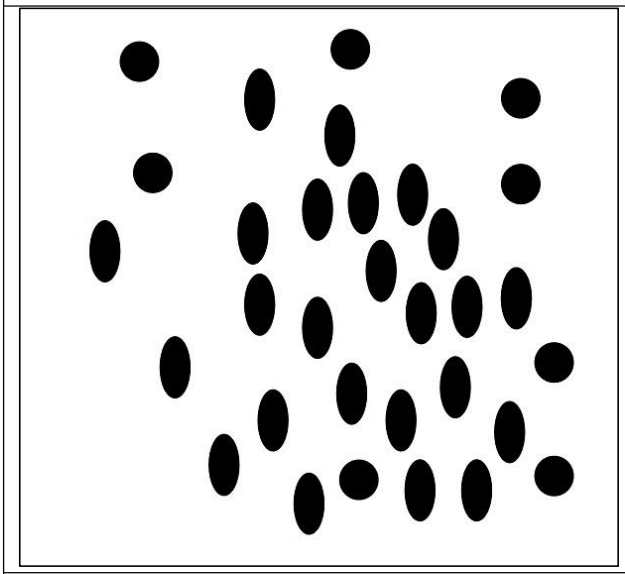
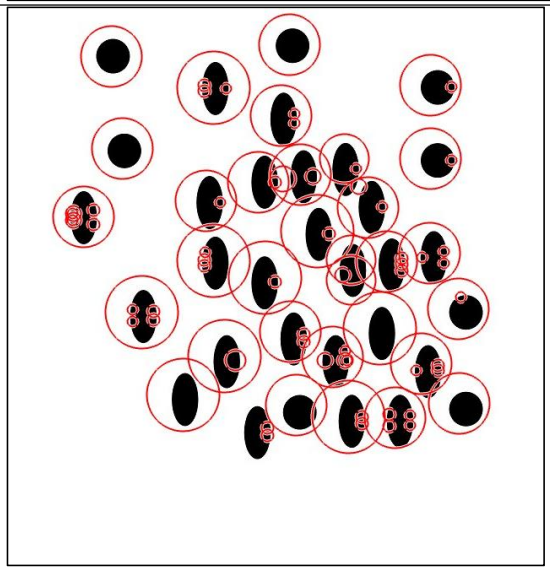
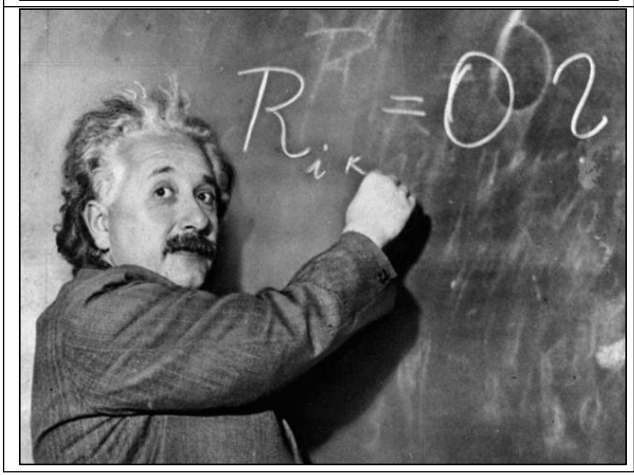
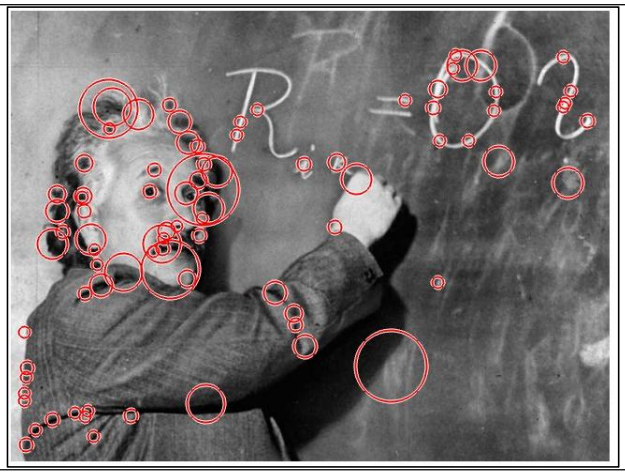
```
 % Rescaling the radius values for input image dimensions
 radius = floor(cols*radius*0.5);

 % Call the viscircles function for drawing circles on the image
 viscircles(blob_xy, radius, 'LineWidth', 1.0);

 % Print the total number of blobs created
 blob_num = sprintf('Total %d blobs are created.', size(blob_xy, 1));
 disp(blob_num)

 % Stop the clock for runtime of the program
 toc
 function [max_x_coord, max_y_coord, max_rad] = detect_blob(img_input, std_dev, in_val,
fin_val, scale_size)
     % Extract the size of image
     [rows, cols] = size(img_input);
     % Create a zeros 3D array to represent scale space
     scale_space_arr = zeros(rows, cols, scale_size);
     % Defining zeros matrix for finding maxima for all 2D scale space
     max_occ = zeros(rows, cols, scale_size);
     % Preallocating empty arrays
     max_x_coord = []; max_y_coord = []; max_rad = []; local_maxima = [];

     % Considering filter size according to Harris combined corner and edge
     % detector and generating scale normalized LoG filter
     log_scaled = (std_dev^2)*fspecial('log', max(1,floor(6*std_dev)), std_dev);
     % Increasing the scale size by 2 for avoiding maxima detection at
     % initial and final steps
     scale_size = scale_size + 2;

     % Calculating the Laplacian of Gaussian (LoG) for change of scaling sizes
     for i=1:scale_size
         % Calculating the factor for scaling the image relative to kernel size
         factor = ((((i-1)*((fin_val-in_val)/(scale_size-1)))/(fin_val-in_val))^2*(fin_val-
in_val)+in_val)/100;
         size_img_resc = max(1,floor(6*std_dev))/factor;
         img_resc_up = imresize(img_input, size_img_resc/cols, 'bicubic');
         % Call function for image convolution and calculate the squared
         % LoG response, resize it to original image size and storing it in
         % space scale 3D array
         scale_space_arr(:, :, i) = imresize((convolution(img_resc_up, log_scaled)).^2, [rows, cols],
'bicubic');
     end
```

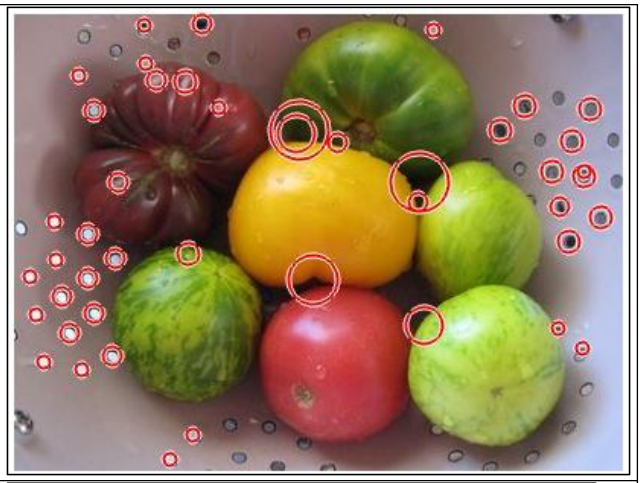
```
    % Iterations for non-max suppression in each 2D slice seperately, for typical Harris
combined corner and edge detector
    for j=1:scale_size
        % Defining threshold value by trial and error (typically around 1000)
        % for image dilation and window size for identifying the maxima
        threshold_val = 1000; win_size = 3;
        % Calculating gray scale dilation for the image
        gray_dil = ordfilt2(scale_space_arr(:, :, j), power(win_size, 2), ones(win_size));
        % Finding and storing maximas for scale space > threshold values
        max_occ(:, :, j) = (scale_space_arr(:,:,j)==gray_dil)&(scale_space_arr(:,:,j)>=threshold_val);
        % Storing rows and cols of the maximas to another matrix
        [row_max, col_max] = find(max_occ(:,:,j));
        % Appending the matrix to store all the row_max and col_max in
        % cell array format
        local_maxima = [local_maxima, {[row_max,col_max]}];
    end

    % After finding non-max suppression in each 2D slice separately, finding
    % the maximas in 3D scale space
    for k=2:(scale_size-1)
        sl_temp = local_maxima(k);
        sl_coord = sl_temp{1};
        sl_coord_size = size(sl_coord);
        % Calculating radius of the blob for the iterative scale_size
        radius = ((((k-1)*((fin_val-in_val)/(scale_size-1)))/(fin_val-in_val))^2*(fin_val-
in_val)+in_val)/100;
        % Rescaling the kernel size for input image dimensions
        kernel_size = floor(cols*radius*0.5);

        % Iterating through the coordinates matrix for every 2D slice
        for p=1:sl_coord_size(1)
            temp_point = sl_coord(p,:);
            x = temp_point(1); y = temp_point(2);
            % For the maximas detected at the edge, we don't want to use
            % them as blob features
            if (x<=kernel_size)||(x>rows-kernel_size)
                continue;
            end
            if (y<=kernel_size)||(y>cols-kernel_size)
                continue;
            end

            % Storing the local maxima in 3D scale space at current
            % location
            local_maxima_slice = scale_space_arr(x,y,k);
```

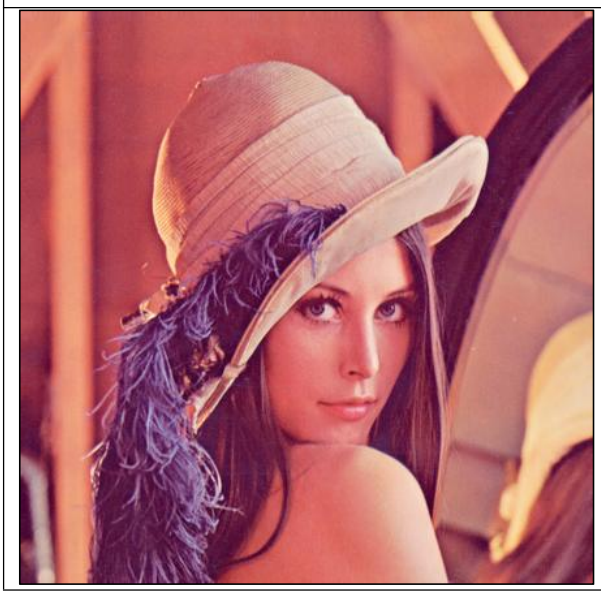
```
            % For window above the current scale space
            win_above = [scale_space_arr(x-kernel_size:x+kernel_size, y-
kernel_size:y+kernel_size, k+1)];
            % For maxima above the current scale space
            max_win_above = max(win_above(:));
            % For window below the current scale space
            win_below = [scale_space_arr(x-kernel_size:x+kernel_size, y-
kernel_size:y+kernel_size, k-1)];
            % For maxima below the current scale space
            max_win_below = max(win_below(:));
            % Finding and storing maximas for current scale space > set
            % coordinates of x,y and radius
            if ((local_maxima_slice > max_win_above) && (local_maxima_slice > max_win_below))
                max_rad = [max_rad, radius];
                max_x_coord = [max_x_coord, x];
                max_y_coord = [max_y_coord, y];
            end
        end
    end
 end

 % Define function for convolution operation for easier referencing
 function [out_conv] = convolution(input_img, kernel)
        % Extracting size of the input image
        [rows, cols, chan] = size(input_img);
        [ker_rows, ker_cols] = size(kernel);
        % Padding the image to implement convolution operation based on the kernel size
        img_pad = zeros(rows+ker_rows-1, cols+ker_cols-1, chan);
        img_pad((ker_rows+1)/2:rows+((ker_rows-1)/2), (ker_cols+1)/2:cols+((ker_cols-1)/2), :)
= input_img(1:rows, 1:cols, :);
        % Extracting the size of padded image and constructing zeros matrix
        % for convolution
        [pad_rows, pad_cols, pad_chan] = size(img_pad);
        out_conv = zeros(pad_rows-ker_rows+1, pad_cols-ker_cols+1, pad_chan);
        % Implementing convolution operation through rows and cols
        for j=1:pad_cols-ker_cols+1
            for i=1:pad_rows-ker_rows+1
                out_conv(i, j, :) = abs(sum((img_pad(i:i+ker_rows-1, j:j+ker_cols-1, :).*kernel),
"all"));
            end
        end
 end
```

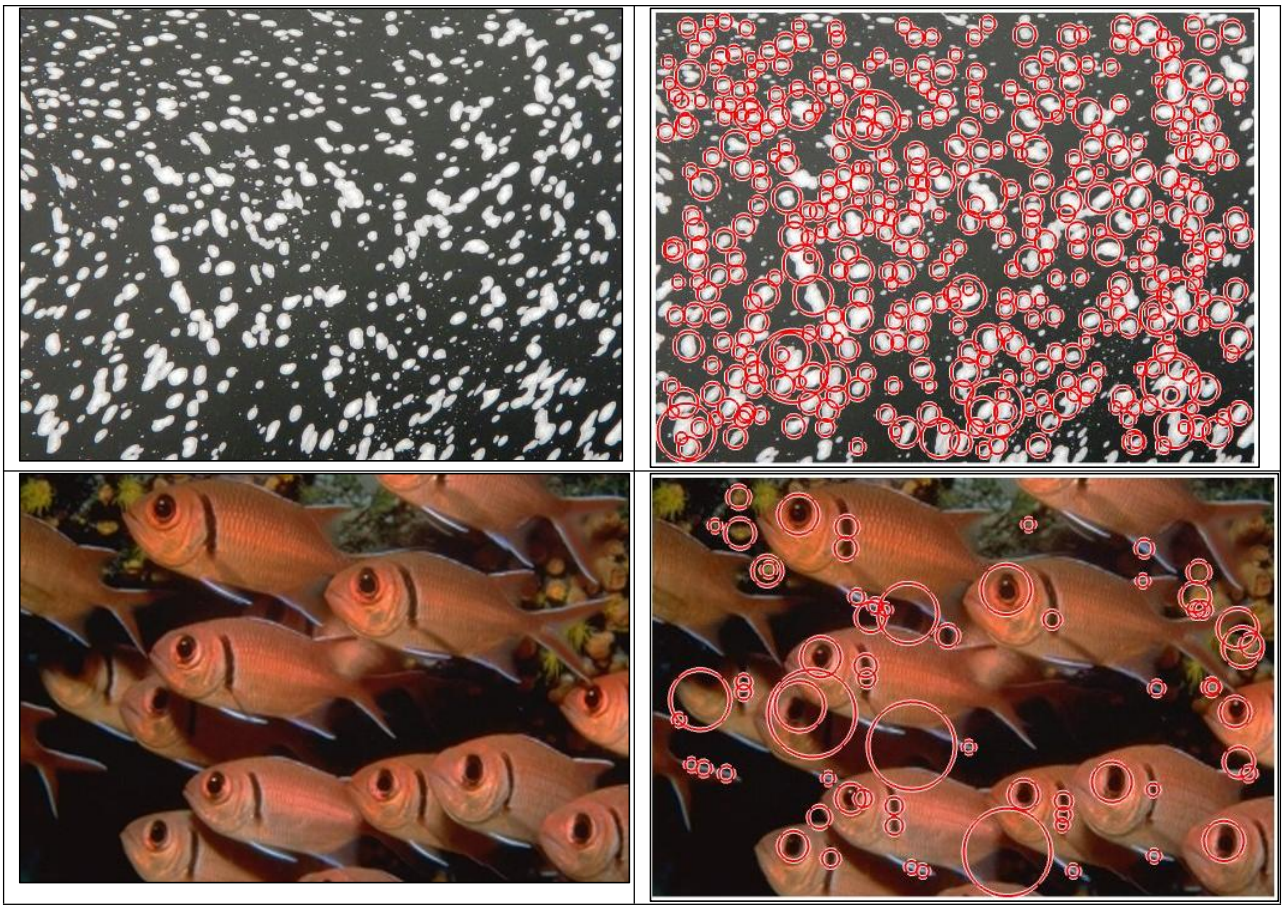
RESULTS:

Test input and output images are saved in ‘awaoo_input_images’ and ‘awaoo_output_images’ folders respectively.

Input Image	Output Image
	
	
	







REFERENCES:

[1] Lecture Notes

[2] Digital Image Processing, Gonzalez and Woods Pearson/Prentice Hall ©

[3] Chris Harris and Mike Stephens. A combined corner and edge detector. In Alvey Vision Conference, Volume 15, pages 10–5244. Manchester, UK, 1988.

[4] Tony Lindeberg. Feature detection with automatic scale selection. International Journal of Computer Vision, 30(2):79–116, 1998.

[5] David G Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91–110, 2004.

[6] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. International Journal of Computer Vision, 60(1):63–86, 2004.

[7] https://en.wikipedia.org/wiki/Blob_detection

[8] https://matlab.fandom.com/wiki/FAQ#How_do_I_create_a_circle.3F

[9] http://www.sci.utah.edu/~weiliu/class/aip/p1/

[10] https://en.wikipedia.org/wiki/Corner_detection

[11] https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf