Project 04

Laplacian Blob Detector

Algorithm outline

1. Generate Laplacian of Gaussian filter
   - The LoG filter is generated using the following equation.

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

   - The function log_kernel() is used to generate the kernel instead of $fspecial$ function of MATLAB so as to get proper zero values in the kernel.
2. Build Laplacian Scale space and go on for n iterations (1 octave and 10 scales chosen in this case)
   - Filter image with scale normalized laplacian at current scale
     - The Laplacian filter generated in the previous step is then multiplied with σ^2 as we need to scale normalize the laplacian. This is because the Laplacian response decreases as we move ahead in the scale. We multiply by σ^2 as the Laplacian is a second derivative.

$$\nabla^2_{norm} g = \sigma^2\left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}\right)$$

     - The generated filter is now used for convolution (Filtering operation) to get a LoG response of the image. The frequency domain (multiplication instead of convolution) filtering is done using the convfreq() function.
     - The convfreq() function takes the filter and the image, does appropriate zero padding to make the filter size equal to the image size, then does appropriate padding to avoid the wraparound error and then finds 2D-DFT of both the image and the filter, then multiplies it and does 2D - IDFT to get the filtered response of the image. *This approach is chosen as it is very fast as compared to convolution in spatial domain.*
     - The number of octaves, initial sigma, filter size dependent on it and the number of scales and the threshold for choosing maximas are all hyper-parameters which are tuned with a heuristic approach.
       * The number of octaves -
       * Initial sigma - From Lowe's SIFT Paper, the initial sigma taken as 1.6 was experimentally found to produce maximum repeatability.
       * Filter size - Heuristically it has been selected as $2 * ceil(3 * sigma\_oct) + 1$ to cover 99.8% of the filter variance.
       * Threshold - The threshold selected will depend on the image and has to be selected heuristically as well. We've found optimal thresholds for images emperically and mentioned them in comments in our code.
   - Save squared Laplacian response at each scale
     - The squared response is then saved to use for the further steps. i.e. Non max suppression.
   - Increase scale progressively by a factor k ($k * \sigma$)
     - Here k is selected to be $\sqrt{2}$. While initial sigma is chosen to be 1.6.
3. Perform non-max suppression in scale space.
   - For non max suppression, we first do 2D non max filtering using ordfilt2() function in MATLAB.

- We do this for every layer in the scale space and then we compare every layer to the layer above and below the current layer to find the actual maximas. This is 3D Non max suppression.
- If it is the true maxima, we store the coordinates in an array of coordinates, max_x, max_y, and also save the corresponding characteristic radius max_r with the same index.
- The characteristic radius is calculated as follows - $r = \sigma * \sqrt{2}$

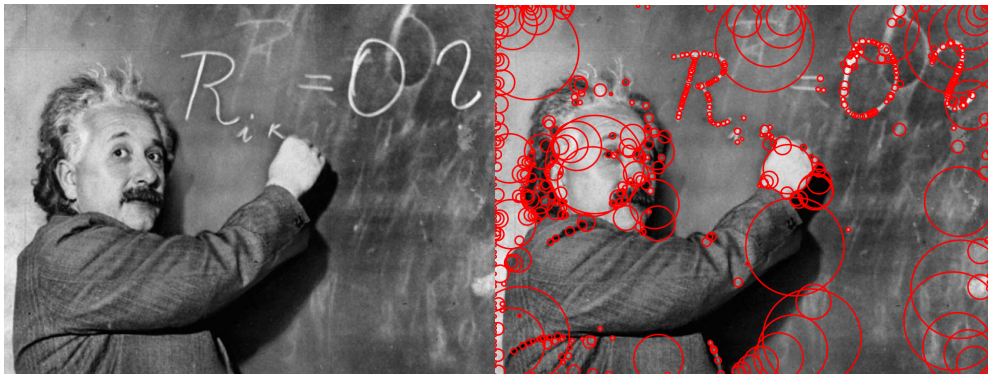
4. Display results at their characteristic scales

- The results are displayed by plotting the circles with their corresponding characteristic radius at their corresponding coordinates (maximas).
- The circs() function is used to display the circles. It uses the parametric equation of the circle $x + rcos(\theta)$ and $y + rsin(\theta)$ to plot the circles with the corresponding radius.

Outputs

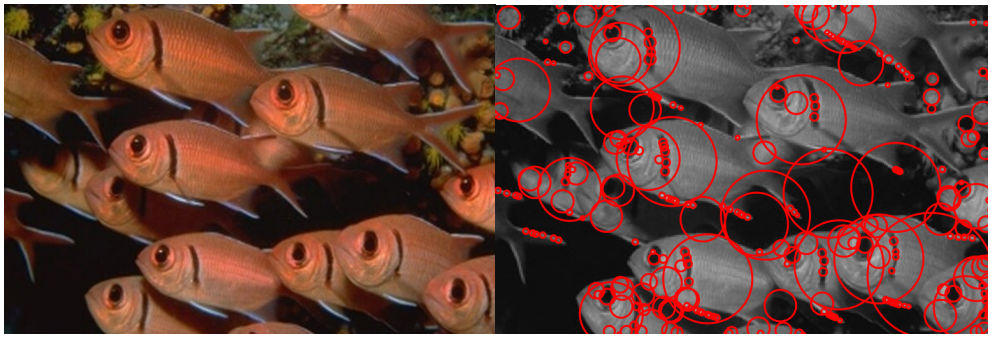
(a) Original Image (b) Detected Blobs

Figure 1: Example 1



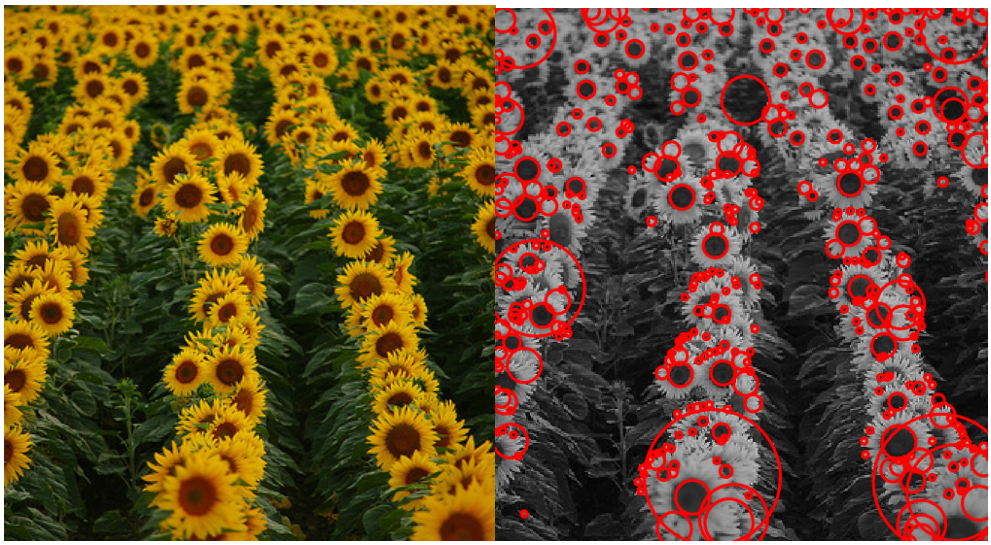
(a) Original Image (b) Detected Blobs

Figure 2: Example 2



(a) Original Image (b) Detected Blobs

Figure 3: Example 3



(a) Original Image (b) Detected Blobs

Figure 4: Example 4

(a) Original Image (b) Detected Blobs

Figure 5: Example 5

(a) Original Image (b) Detected Blobs

Figure 6: Example 6

(a) Original Image (b) Detected Blobs

Figure 7: Example 7

(a) Original Image (b) Detected Blobs

Figure 8: Example 1

Appendix/ Codes

```
clc; close all;
%-------------------------------------------------------------------------%
% LAPLACIAN BLOB DETECTION
% Authors - Mayuresh Sardesai, Dinesh Bhosale, Omkar Mujumdar
% 1. Generate Laplacian Scale space using the generate log filter
% 2. Perform non max suppression
% 3. Plot the resulting circles at their corresponding scales
%-------------------------------------------------------------------------%
tic;
image = imread('TestImages4Project\simba.jpg');
grayImg = im2double(rgb2gray(image));
sigma=1.4; %initial sigma
k=sqrt(2);  %change sigma for every iteration by a factor of sqrt(2)
octaves = 1;    %set number of octaves
% # Scales for images
% 1. Butterfly - 10
% 2. Einstein - 10
% 3. Fishes - 10
% 4. Sunflowers - 10
% 5. Lena - 10
% 6. Omkar - 10
% 7. Eye - 10
% 8. Simba - 10
intervals = 10; %set number of itervals in a one scale
%Threshold for images
% 1. Butterfly - 0.02
% 2. Einstein - 0.03
% 3. Fishes - 0.0175
% 4. Sunflowers - 0.0325
% 5. Lena - 0.025
% 6. Omkar - 0.025
% 7. Eye - 0.025
% 8. Simba - 0.02
thresh = 0.02;
%empty arrays for storing the coordinates and the corresponding radius
max_x = [];
max_y = [];
max_r = [];
l = cell(octaves,1);    %empty cell array to store the scale space
fsize = cell(octaves,1);%empty cell array for storing filter sizes at each oct
sig = cell(octaves,1);  %empty cell array to store sigma values at each oct
sigma_op = sigma;       %init sigma_op - sigma at each level of scale space
sigma_oct = sigma;      %init sigma_oct - initial sigma at each octave
for oct = 1:octaves    %traverse through octaves
    [rows,columns]=size(grayImg);   %size of the grayscale image
    sigma_op = sigma_oct;   %set sigma_op as the sigma at the current octave
    for i=1:intervals+1     %traverse through the number of scales
        fsize{oct}(i) = 2*ceil(3*sigma_op)+1;   %calculate filter size
        sig{oct}(i) = sigma_op; %store sigma_op
        %generate log filter in spatial domain based on the formula
        filter = (sigma_op.^2)*log_kernel((2*ceil(3*sigma_op)+1),sigma_op);
        l{oct}(:,: ,i) = convfreq(grayImg, filter); %get log response
        l{oct}(:,:,i) = l{oct}(:,:,i).^2;   %get squared response
%         figure;
%         imshow(l{oct}(:,:,i));
        sigma_op=sigma_op*(k);  %update sigma_op by multiplying it with k
    end
    sigma_oct = sigma*(2^oct);  %update sigma_oct
    %downscale image and smoothen with gaussian having sigma_oct
    grayImg = convfreq(downscale(grayImg,0.5), ...
        genKernel('gaussian', (2*ceil(3*sigma_oct)+1), sigma_oct));
    %resize the image back to the original size.
```

```
    grayImg = upscale(grayImg, [rows columns]);
    %perform non max suppression
    [max_x, max_y, max_r] = nonmaxs(l{oct}, sig{oct}, ...
        intervals, max_x, max_y, max_r, thresh);
%     [max_x, max_y, max_r] = nms(l{oct}, sig{oct}, fsize{oct}, ...
%         intervals, max_x, max_y, max_r);
end
figure;
imshow(rgb2gray(image)); hold on;   %display the image
%plot the circles
for i = 1:length(max_x)
%     viscircles([max_y(i), max_x(i)], max_r(i));
    circs(max_x(i), max_y(i), max_r(i));
end
toc
saveas(gcf, 'Output.png');
```

Code 1: Main Blob Detection

```
function [max_x, max_y, max_r] = nonmaxs(currScale,sig, intervals, max_x, max_y, max_r, thresh)
%NMS Non max suppression
%   NMS - first find maximas in each layer. And then compare maximas with
%   the maximas from the layer above and below the particular layer. (3d
%   nms)
    mx = zeros(size(currScale,1), size(currScale,2),intervals);
%     for 2D non max filtering
    for i = 1:intervals+1
        mask_size = 3;
        %max filtering using a 3x3 window
        mx(:,:,i) = ordfilt2(currScale(:,:,i), mask_size^2, ones(mask_size));
    end
%     3D non max filtering
    for i = 1:intervals+1
%         consider the maximum among the 3 adjacent layers
        mx(:,:,i) = max(mx(:,:,max(1,i-1):min(intervals+1,i+1)),[],3);
    end
    mx = mx.*(mx==currScale); %getting maximas
    for i = 1:intervals+1
        curr = mx(:,:,i);   %maximas at current scale
        [r,c] = find(curr>thresh);  %discard maximas below threshold
        %can be used to eliminate blobs at the edges.
%         rd = r(r>=1+sig(i)*sqrt(2) & r<=max(r)-sig(i)*sqrt(2) & c>=1+sig(i)*sqrt(2) ...
%             & c<=max(c)-sig(i)*sqrt(2));
%         cd = c(r>=1+sig(i)*sqrt(2) & r<=max(r)-sig(i)*sqrt(2) ...
%             & c>=1+sig(i)*sqrt(2) & c<=max(c)-sig(i)*sqrt(2));

%         repeat radius as a vector for all maximas at current scale
        current_radius = repmat(sig(i)*sqrt(2),[length(r) 1])';
        max_x = [max_x, [r]'];
        max_y = [max_y, [c]'];
        max_r = [max_r, [current_radius]];
    end
end
```

Code 2: Non Max Suppression

```
function [respo] = convfreq(img,kern)
%CONVFREQ Convolution in frequency domain (Frequency domain filtering)
%   Filter the image in frequency domain. Use the dft2 and idft2 functions
%   for conversion to frequency domain.
    [rk,ck]=size(kern); %kernel size
    [r,c] = size(img);  %image size
    if ~(r>rk && c>ck)  %calculate padding based on kernel size
        zpr = r+2*floor(rk/2); zpc = c+2*floor(ck/2);
```

```
    else   %dont pad if kernel size is smaller than image size
        zpr = r; zpc = c;
    end
    imgs = zeros(zpr,zpc);
    imgs(floor(zpr/2)+(1:r)-floor(r/2), floor(zpc/2)+(1:c)-floor ...
        (c/2),:) = img; %zero pad image
    imgs = pad4dft(imgs); %padding to avoid wraparound error
    imgfreq = dft2(fftshift(imgs)); %calculate dft of image
    [r,c] = size(imgs); %get size of new padded image
    zpr = r; zpc = c;
    kernfreq = zeros(zpr,zpc);  %pad kernel
    kernfreq(floor(zpr/2)+(1:rk)-floor(rk/2), floor(zpc/2)+(1:ck)-floor ...
        (ck/2),:) = kern;   %pad kernel
    kernfreq = dft2(kernfreq);  %calculate dft of kernel
    opfreq = imgfreq.*kernfreq; %calculate the output in frequency domain
    resp = real(idft2(opfreq)); %consider real part of idft
    respo = ifftshift(resp);    %shift the output
    respo = respo(end/2+1:end, end/2+1:end);    %'unpad' image
    if ~(size(img,1)>rk && size(img,2)>ck)  %'unpad' image
        respo = respo(1+floor(rk/2):end-floor(rk/2),1+floor(ck/2):end-floor(ck/2));
    else
        respo = respo;
    end
end
```

Code 3: convfreq - freq domain filtering

```
function [fftImg] = dft2(img)
%DFT2 Summary of this function goes here
%   Detailed explanation goes here
%     img = (img-min(img(:)))./(max(img(:))-min(img(:)));
    %approach 1 - operate fft on first dimension first and then the second
    %using the syntax fft(x, [], dim)
    [r, c ,ch] = size(img);
    if ch>1
        fftImg(:,:,1) = fft(fft(img(:,:,1), [],1), [], 2);
        fftImg(:,:,2) = fft(fft(img(:,:,2), [],1), [], 2);
        fftImg(:,:,3) = fft(fft(img(:,:,3), [],1), [], 2);
    else
        fftImg = fft(fft(img, [],1), [], 2);
    end
%     Approach 2 - use 2 for loops, one to take fft of the rows. Then use
%     another one to take the fft of the obtained result from the first
%     loop
%     [r,c,ch] = size(img);
%     for c = 1:ch
%         for i = 1:r
%             fftImg(i,:,ch) = fft(img(i,:,ch));
%         end
%         for j = 1:c
%             fftImg(:,j,ch) = fft(fftImg(:,j,ch));
%         end
%     end
end
```

Code 4: 2D - DFT - calculate DFT using 1D FFT

```
function [recovImg] = idft2(dftImg)
%IDFT2 Summary of this function goes here
    [M N ch] = size(dftImg);
    if ch>1
        recovImg(:,:,1) = (1/numel(dftImg(:,:,1))).*real(conj(dft2(conj(dftImg(:,:,1)))));
        recovImg(:,:,2) = (1/numel(dftImg(:,:,2))).*real(conj(dft2(conj(dftImg(:,:,2)))));
        recovImg(:,:,3) = (1/numel(dftImg(:,:,3))).*real(conj(dft2(conj(dftImg(:,:,3)))));
```

```
        recovImg(:,:,1) = (recovImg(:,:,1)-min(recovImg(:,:,1)))./(max(recovImg(:,:,1))-min(recovImg(:,:,1)));
        recovImg(:,:,2) = (recovImg(:,:,2)-min(recovImg(:,:,2)))./(max(recovImg(:,:,2))-min(recovImg(:,:,2)));
        recovImg(:,:,3) = (recovImg(:,:,3)-min(recovImg(:,:,3)))./(max(recovImg(:,:,3))-min(recovImg(:,:,3)));
    else
        recovImg = (1/numel(dftImg)).*real(conj(dft2(conj(dftImg))));
%         recovImg = (recovImg-min(recovImg(:)))./(max(recovImg(:))-min(recovImg(:)));
    end
end
```

Code 5: 2D - IDFT - calulate idft using 1D FFT

```
function log_kernel = log_kernel(siz,sig)

if  mod(siz,2)==0
    xcenter=(siz/2)-1;
    ycenter=(siz/2)-1;
else
    xcenter=floor(siz/2);
    ycenter=floor(siz/2);
end

for i=-xcenter:floor(siz/2)
    for j=-ycenter:floor(siz/2)
        log_kernel(i+xcenter+1,j+ycenter+1)=(1/(pi*(sig^4)))*((((i^2)+(j^2))/(2*(sig^2)))-1)*(exp(-((i^2)+(j^2))/(2*(sig^2))));
    end
end
```

Code 6: log_kernel - generate log kernel

```
function [padImg] = pad4dft(img)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
    padImg = zeros(2*size(img,1:2));
    padImg(1:size(img,1), 1:size(img, 2))=img;
end
```

Code 7: pad4dft - padding for dft

```
function [ResampledImage] = upscale(InputImage,newSize)
%UPSCALE used to upscale image to selected size
%   newSize taken as input to avoid mismatch of dimensions while moving
%   through the pyramid.
oldSize = size(InputImage);
scale = newSize./oldSize;

% nearest neighbor interpolation.
% calculate the new row and column indices for assignment of pixels
%subtract -0.5 to consider center of pixel. Add 0.5 to negate subtraction
%min function used to avoid any overflows
rowIndices = min(round(((1:newSize(1))-0.5)./scale(1)+0.5), oldSize(1));
colIndices = min(round(((1:newSize(2))-0.5)./scale(2)+0.5), oldSize(2));
%assignment based on the newly calculated indices
ResampledImage = InputImage(rowIndices, colIndices,:);
end
```

Code 8: upscale

```
function [ResampledImage] = downscale(InputImage, ScalingFactor)
%IMRESAMPLE Resample image according to the given input
```

```
%   Resize the given image according to the given scaling factor, implement
%   nearest neighbor interpolation while upsampling to sample the images.

scale = [ScalingFactor, ScalingFactor];
oldSize = size(InputImage);
newSize = max(floor(scale.*oldSize(1:2)),1); %to avoid 0 index error
% nearest neighbor interpolation.
% calculate the new row and column indices for assignment of pixels
rowIndices = min(round(((1:newSize(1))-0.5)./scale(1)+0.5), oldSize(1));
colIndices = min(round(((1:newSize(2))-0.5)./scale(2)+0.5), oldSize(2));
ResampledImage = InputImage(rowIndices, colIndices,:);
end
```

Code 9: downscale

```
function [h] = circs(x,y,r)
%CIRCS Plots circles at given center and radius
%   Uses the parametric equation of the circle and
    hold on
    th = linspace(0,2*pi, 150);
    xunit = r * cos(th) + x;
    yunit = r * sin(th) + y;
    h = plot(yunit, xunit, 'r', 'LineWidth', 1.5);
    hold off
end
```

Code 10: circs - draw circles at specified coordinates and radius