Table of Contents

.... 1
This function evaluates the most stable Harris corner points at different scales based on Laplacian of Gaussian (LOG) formulation using Scale pyramid 1
References: 1
Inputs: 1
Outputs: 2
This is the main loop which runs over all the images. 2
Part #1 , Finding Harris-corner points by thresholding **Harris measure,R** 2
Part #2 , Checking the stability of intereset points using Laplacian of Gaussian (LOG) 4


```
function my_final = Harris_Laplace_fn(Img, threshold_R)
```

This function evaluates the most stable Harris corner points at different scales based on Laplacian of Gaussian (LOG) formulation using Scale pyramid

Written by : Sameer Sharma (ED13D007), Engg. design , IITM

References:

1. Harris-Affine & Hessian Affine: K. Mikolajczyk and C. Schmid, Scale and Affine invariant interest point detectors. In IJC V 60(1):63-86, 2004.

2. Digital Video Processing (http://10.6.4.152/dvp/dvp.html), Computer Sc., Indian Institute of Technology, Madras.

3. A Comparison of Affine Region Detectors, K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, L. Van Gool. IJCV 2005.

4. Scale-Space Theory in Computer Vision, T. Lindeberg. Springer

5. http://en.wikipedia.org/wiki/Corner_detection
The_Harris_Stephens_Plessey_Shi_Tomasi_corner_detection_algorithm

6. Local Invariant Feature Detectors - A Survey, T. Tuytelaars and K. Mikolajczyk. FGV 2008

Inputs:

```
*Img* : 'String datatype of the current image name ( If colored,
convert to gray scale first)
*threshold_R* : Threshold on the Harris measure for corner
detection ( Det(M) - k* (trace(M))^2 )
```

Outputs:

```
*my_final* : a N x 3 array with spatial cordinates and scales of
the computed Harris interest points
```

Defining scale-levels and initial-scale(these can be further customized)

```
levels = 12;
sigma_initial = 1.2;
scale_factor = 1.2;
myPoints = [];
my_final = [];
```

This is the main loop which runs over all the images.

The loop is divided into segments such that it will be easy for the user to track the process.

```
for n = 1:levels
```

Defining the current scale (Sigma value for the Guassian mask)

```
sigma = (scale_factor)^n*sigma_initial;
```

Part #1 , Finding Harris-corner points by thresholding Harris measure,R

Define a Guassian filter mask for a given value of sigma scales Making sure that the filter size is ODD with pixel of interest at the center

```
patch_size = 6*ceil(sigma)+1;
if mod(patch_size,2) ==0;
    patch_size = patch_size +1;
end
```

Mask definition

```
mask = fspecial('gaussian',patch_size,sigma);
```

Gaussian differentitation (0.7 * sigma)

```
G1 = fspecial('gaussian',patch_size,0.7*sigma);
[Gx,Gy] = gradient(G1);
```

Computing second order derivatives for later use(Laplacian computation)

```
[Gxx,~] = gradient(Gx);
[~,Gyy] = gradient(Gy);
myImage1 = double((imread(Img)));
myImage = filter2(mask, myImage1);
```

Compute the gradients of the Gaussians

```
[ro, col] = size(myImage);
grad_x = filter2(Gx, myImage1);
grad_y = filter2(Gy, myImage1);
grad_xx = filter2(Gxx, myImage1);
grad_yy = filter2(Gyy, myImage1);
```

Assembling the Moment matrix (2 X 2)

```
M1 = grad_x.^2;
M2 = grad_y.^2;
M3 = grad_x.*grad_y;
```

Mask the three matrices using Guassian window

```
M1_new = filter2(mask, M1);
M2_new = filter2(mask, M2);
M3_new = filter2(mask, M3);
R = zeros(ro, col);
```

Computing the equivalent to the eigen values of the moment matrix for each pixel by defining the measure R ***(Szieliski) : det(M)/trace(M)*** or **Harris = det(M) - K*trace(M)^2**, K can be taken as 0.7

```
R = (M1_new.*M2_new - M3_new.^2)./(M1_new + M2_new);
```

Normalize the R matrix such that the values lies [0 % , 100 %]

```
R_norm = 100*(R - min(min(R)))/(max(max(R)) - min(min(R)));
```

Thresholding based on the normalised R value

```
myLogic = R_norm > threshold_R;
% figure, imshow(myLogic);
```

Padding the boudaries

```
myLogic(1: ceil(patch_size/2),:) = 0;
myLogic(:,1: ceil(patch_size/2)) = 0;
myLogic(end-ceil(patch_size/2):end,:) = 0;
myLogic(:,end- ceil(patch_size/2):end) = 0;
R_norm_new = R_norm.*double(myLogic);
bound = floor(patch_size/2);
```

Check the *interest point*, if it has attained maximum value of R as compared to the neigbours i_z and index are padded to avoid boundary problems

```
for i_z = bound + 1:ro - bound
```

Scan row-wise of the thresholded masked Non-Zero R values

```
    index = find(R_norm_new(i_z,:));
```

Check if we have an interset point at that index

```
    size_new = size(index,2);
```

Check if the intereset point has maximum R value as compared to the neighbours

```
    if ~isempty(index) && min(index)> bound && max(index)<col - bound
```

```
        for check_index = 1:size_new

        neigbors = R_norm_new(i_z - bound:i_z +bound,...
               index(check_index)- bound:index(check_index)+...
              bound);
         [~, colIdx] = max(max(neigbors,[],1));
         [~, rowIdx] = max(max(neigbors,[],2));
```

Update the final mask if it is maximum

```
            if rowIdx*colIdx ~= (bound + 1)^2

            myLogic(i_z,index(check_index)) = 0;
            R_norm_new(i_z,index(check_index)) = 0;
            end

        end


    end

end
```

Part #2 , Checking the stability of intereset points using Laplacian of Gaussian (LOG)

Choosing the neighbours and assembling corner points for all the scales

```
[ro1, col1] = find(myLogic);
```

LOG of the image at a scale

```
LOG{n} = (sigma^2)*sqrt(grad_xx.^2 + grad_yy.^2);
L_current = LOG{n};
```

Assembling all the potential interset points (outputs of R threshold) in an array (3 X N)

```
index_log = ro1 + (col1 - 1)*size(myLogic,1);
myPoints{n} = [ro1, col1, sigma*ones(numel(ro1),1)];
% figure, imshow(uint8(myImage));hold on,plot(col1, ro1,'+');
```

Maximizing the LOG and choosing the final HARRIS points. for the two extreme scales (Top and Bottom) we only have one neighbor to compare with, but for rest of the scales compare LOG accross three scales.

Uppermost scale

```
if n == 2

    L = LOG{1};
    myP = myPoints{1};
    index_check = myP(:,1) + (myP(:,2) - 1)*size(myLogic,1);
    diff_2 = L(index_check) - L_current(index_check);
```

Check for the maximum range

```
    for check = 1: size(myPoints{1},1)
        if diff_2(check)>0
            my_final = [my_final ; myP(check,1),myP(check,2),myP(check,3)];
        end
    end
```

All the middle range scales

```
elseif n>2 && n<levels

    L_center = LOG{n-1};
    L_up = LOG{n-2};
    myP = myPoints{n-1};
    index_check = myP(:,1) + (myP(:,2) - 1)*size(myLogic,1);
    diff_down = L_center(index_check) - L_current(index_check);
    diff_up = L_center(index_check) - L_up(index_check);
     for check = 1: size(myP,1)
        if diff_down(check)>0 && diff_up(check) >0
            my_final = [my_final ; myP(check,1),myP(check,2),myP(check,3)];
        end
    end
```

Bottom scale

```
elseif n==levels
        L_up = LOG{n-1};
        myP = myPoints{10};
        index_check = myP(:,1) + (myP(:,2) - 1)*size(myLogic,1);
        diff_10 = L_current(index_check) - L_up(index_check);
    for check = 1: size(myP,1)
         if diff_10(check)>0
            my_final = [my_final ; myP(check,1),myP(check,2),myP(check,3)];
        end
    end

end

end
% figure, imshow(uint8(imread(Img)));hold on;
% plot(my_final(:,2), my_final(:,1),'r*');
```

Published with MATLAB® R2014a