Project 1 Content Based Image Retrieval Using Global and Local Features

Daniel Bryan and Olmo Zavala
Florida State University

October 19, 2014

For this project three approaches for Content Based Image Retrieval (CBIR) where implemented. The first approach is based on the comparison of color histograms, the second solution uses spectral histogram to do the comparison of images, and the third approach uses SIFT features to compare images.

1 Description

In this section the three methods used for CBIR are described.

1.1 Color histogram

In this method we used the color (intensity filter) histogram to compare images. The following steps summarizes the method:

1. **Image pyramidization.** The images sizes where reduced to half in order to reduce the time of the algorithm. The method used to compute the pyramids is *Gaussian pyramid*. The first step of the Gaussian pyramid algorithm is to blur the image using a Gaussian filter, and then scale the image down by creating one pixel from the average color of four pixels.

2. **Compute histograms.** The number of bins used for this method is 256, for each color band.

3. **Compute histogram distances.** The distance between each pair of image histograms was computed using histogram intersection:

$$dist(hist_a, hist_b) = \sum_{i=1}^{256} \min(hist_a(i), hist_b(i)) \tag{1}$$

4. **Images similarity**. The distance between the histograms of the images was used as the *similarity* parameter.

1.2 Spectral histogram

This method uses spectral histograms for CBIR. The proposed steps for this method are as follows:

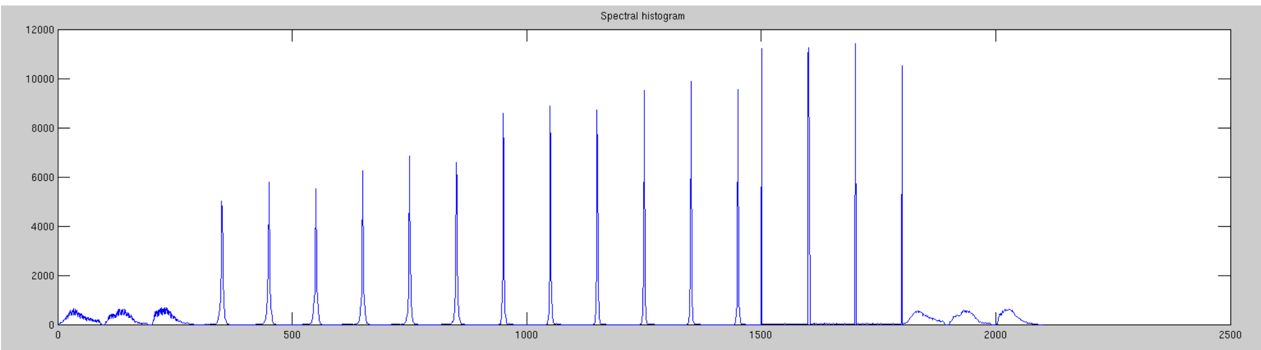
1. **Image pyramidization.** The images sizes where reduced to half in order to reduce the time of the algorithm. The method used to compute the pyramids is *Gaussian pyramid*. The first step of the Gaussian pyramid algorithm is to blur the image using a Gaussian filter, and then scale the image down by creating one pixel from the average color of four pixels.

2. **Filter images**. Six filters in addition to the intensity filter are applied to each of the images. The filters used and their corresponding masks are:

$$\begin{aligned}
\frac{\partial I}{\partial x} &= [0 \;\; -1 \;\; 1] \\
\frac{\partial I}{\partial y} &= [0 \;\; -1 \;\; 1]^T \\
\frac{\partial I}{\partial x \partial x} &= [-1 \;\; 2 \;\; -1] \\
\frac{\partial I}{\partial y \partial y} &= [-1 \;\; 2 \;\; -1]^T \\
LoG(I(x,y)) &= (x^2 + y^2 - \sqrt{2\sigma^2})e^{-(x^2+y^2)/\sqrt{2\sigma^2}} \\
Gauss(I(x,y)) &= \frac{1}{2\pi\sigma^2} e^{\frac{-(x.^2+y.^2)}{2\sigma^2}}
\end{aligned} \tag{2}$$

   For the Laplacian of Gaussian (LoG) and Gaussian filters the size of the filter used is 5 with a sigma value of 0.7.

3. **Compute spectral histograms.** The number of bins used for this method is 100, for each color band and for each filter. For all of the filters the range goes from [0 256], even when the values of some of the filters may go from [-256 256]. Several options for the range of the histograms were tested, and the range of [0 256] gave the best results. The 100 bins were evenly split from [0 256]. Figure 3 shows an example of the spectral histograms.



4. **Compute histogram distances.** The distance between each pair of image histograms was computed using histogram intersection:

$$dist(hist_a, hist_b) = \sum_{i=1}^{100} \min(hist_a(i), hist_b(i)) \tag{3}$$

5. **Images similarity**. The distance between the histograms of the images was used as the *similarity* parameter.

1.3 Scale-Invariant Feature Transform (SIFT)

For this method we used the SIFT descriptors to compare images. The code is based on the implementation provided at http://www.vlfeat.org/. The proposed steps to obtain a score that could be used for CBIR are as follows:

1. Obtain the SIFT descriptors for each image using function **vl_sift**.
2. Compute the matching descriptors and the distance between those descriptors using the function **vl_ubcmatch**.
3. Obtain the maximum distance obtained between all the matches (for later normalization).
4. Combine the average distance of the matched descriptors with the number of matched descriptors between two images using *Cantor pairing*. Equation 4 shows the score computed between images i and j using the SIFT features.

$$\begin{aligned} k1 &= mean\left(\frac{\text{matches}(i,j)}{\text{maxDist}}\right) \\ k2 &= \text{length}(\text{matches}(i,j)) \\ \text{siftScore}(i,j) &= (0.5 * (k1 + k2) * (k1 + k2 + 1)) + k2 \end{aligned} \tag{4}$$

1.4 Computation time

The three methods were implemented in parallel (when possible) using the matlab function $parfor$. The third method, which uses SIFT features, takes significantly more time and is presented separately at the end of this section.

For efficiency, the first two methods are implemented in the same code, but the program *Problem1And2CompTime* can be easily modified to display the time that takes for each method to compute the CBIR. The times shown in table 1 are for a specific run on a machine running Ubuntu 14.04, with matlab 2012a, with an i7 intel processor and 10 GB of ram. The times varies a little for each run but table 1 gives a good approximation of the time that each method takes.

	Color Hist (sec)	**Spectral Hist (sec)**
Read images, filter and compute histogram	9.38	30.01
Calculate distances	6.69	10.78
Compute Precision Recall	1.47	1.45
Avg. Precision and Rank	1.22	1.34
Total time	**17.54**	**43.58**

Table 1: Computation time for **Color histogram** and **Spectral histogram** methods.

The method based on SIFT features takes significantly more time to complete. In order to debug our proposed method, the computation of the SIFT features and descriptors was run once and then saved in a separate file (*/Variables/descriptors.mat*). The results of the descriptor matching, which takes almost two hours to finish, were also saved in a separate file (*/Variables/scores.mat*). In this way we only had two compute the matching and the descriptors once for the image set. The times shown in table 2 are also obtained from a Laptop computer running Ubuntu 14.04, with matlab 2012a, with i7 intel processor and 10 GB of ram.

	SIFT (time sec)
Read all Images	12.47
Detect SIFT descriptors	83.3
Matching descriptors	approx. 7 sec/image $\sim$ 7000 sec $\sim$ 1:56 hrs
SIFT score	18.8
Precision recall	1.6
Average PR	1.44
Total	**$\sim$ 2 hrs**

Table 2: Computation time for **SIFT** features method

2 Results

To better compare the results between the proposed methods, the results were plotted together. Figure 1 displays the average precision (left) and average rank (right) for each of the ten classes of images. There are four methods analyzed: SIMPLIcity (red star), color histogram (green circle), spectral histogram (blue square), and SIFT features (magenta diamond).

The highest performing of the proposed methods is the **Spectral Histogram**. It improves the results shown in the SIMPLIcity paper in 8 of the 10 categories for the average precision and in 2 of the 10 categories for the average rank. The **Color histogram** method obtained higher average precision for some of the categories (1, 8, and 10), but in general is a little bit behind than the Spectral histogram method. The **SIFT** features method seems to over fit the matching of the images and in general obtained poor results. This method is still the only one that surpasses SIMPLIcity for the 3rd category in the average precision evaluation. We believe that the problem of the method used compare images based on the SIFT features is that it is not taking into account a 'general' similarity between the descriptors. The function that was used (vl_ubmatch) computes the closest descriptors and their distances, but there is no indication that images in the same class consistently share similar features. A better approach may be to match only the most representative features between the images, which may require semantic interpretation.

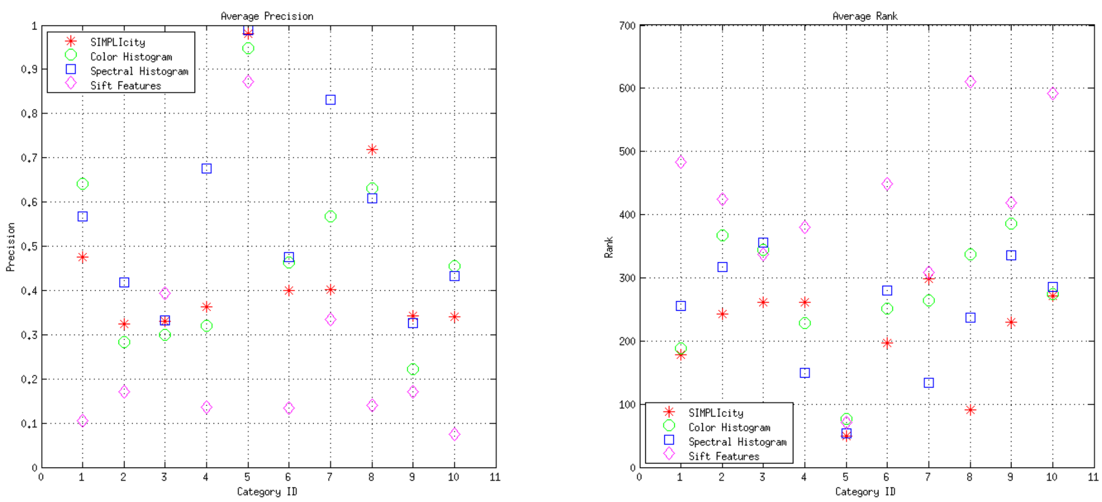


Figure 1: Average PR and rank for our three proposed methods.

The following are good and bad examples of Precision/Recall curves (PR) obtained for each methods. Figure 2 shows the PR obtained for image #401 using color histogram and spectral histogram methods. In this case both methods obtained good PR curves.

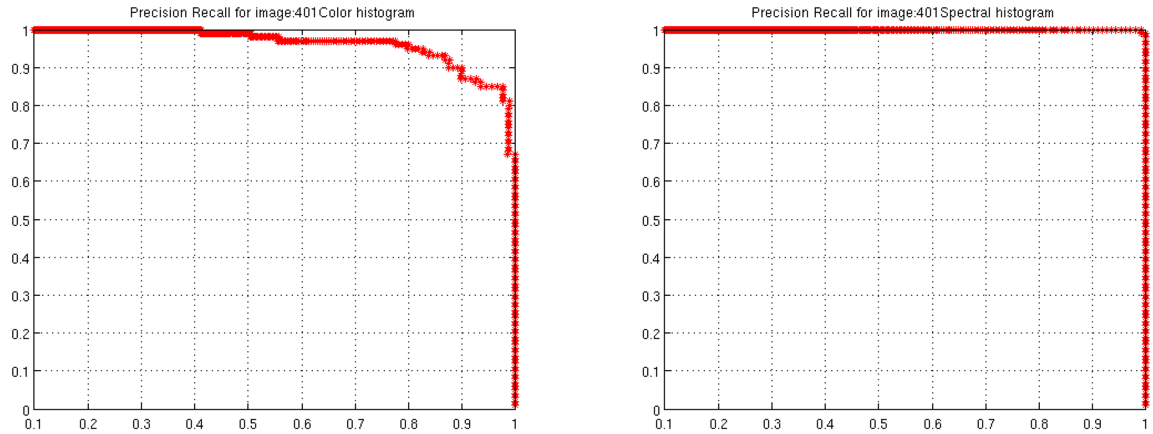


Figure 2: Precision recall results using color and spectral histograms for image 401. Example of good results.

Figure 3 shows the PR obtained for image #361 color histogram and spectral histogram methods. In this case both methods obtained poor PR curves.

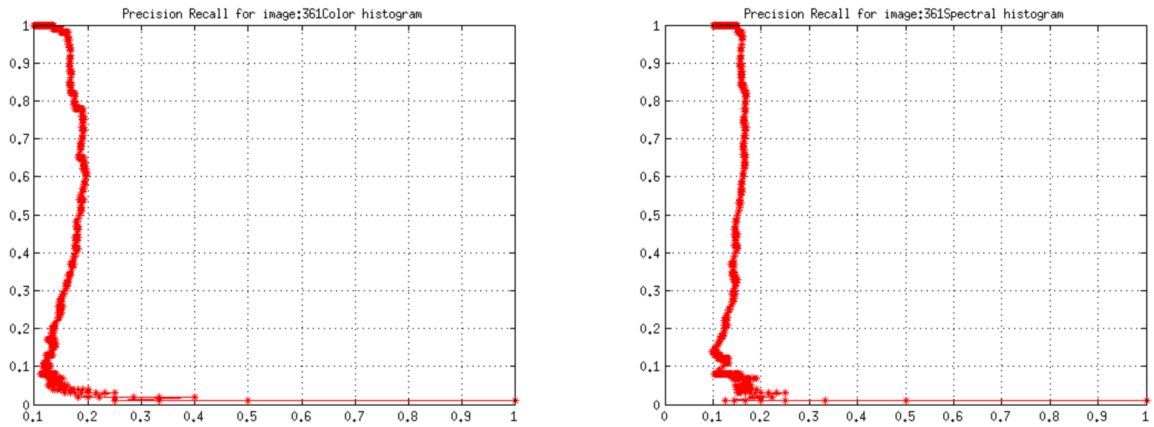


Figure 3: Precision recall results using color and spectral histograms for image 361. Example of bad results.

Finally, figure 4 shows the PR curves obtained using the SIFT features method. The first case is the image #481 which obtains very good results. The right plot of figure 4 shows the PR curve for image #331 where it obtains very poor results.

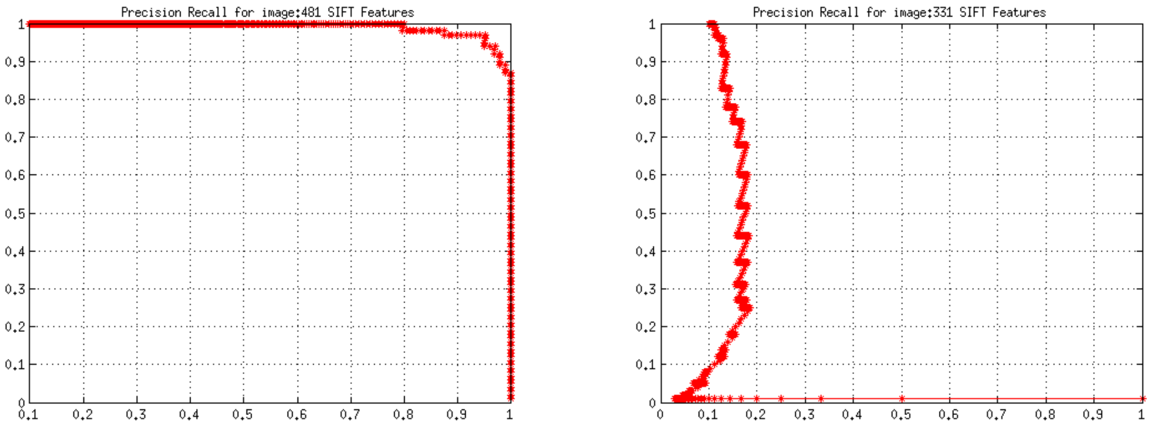


Figure 4: Precision recall plot for one good (image 481) and one bad (image 331) case, using the SIFT features methods.

3 Code

Table of Contents


```
clc;
clear all;
close all;
addpath(genpath('externalLib'));
addpath(genpath('Variables'));

%histograms
totalImages = 1000;

% Running in parallel, check if the pool of threads is already open
if matlabpool('size') == 0
    infoLocal = parcluster('local');
    maxWorkers = infoLocal.NumWorkers;
    matlabpool('open',maxWorkers);
end

bins = [256 100];% Number of bins for each problem (1 and 2)

imgHists = zeros(totalImages, bins(1)*3); % Histograms for problem 1
imgSpectralHists = zeros(totalImages, bins(2)*7*3); % Histograms for problem 2 (us

display('Reading images and computing the histograms....');
tic;
parfor_progress(totalImages);
parfor currIndx =1:totalImages
    % Read images
    fname=sprintf('corel/%i.jpg',currIndx-1);
    currImgInt = imread(fname,'jpg');

    % Reducing size of image
    currImg = double(impyramid(currImgInt,'reduce'));

    [filteredImg numFilters]= filterImages(currImg,3,1);% Filtered image for probl
    %imshow(uint8(squeeze(filteredImg(1,:,:,1))));

    imgHists(currIndx,:) = computeHist(1, currImg, 1, bins(1));
    %plot(imgHists(currIndx,:));
    imgSpectralHists(currIndx,:) = computeHist(1, filteredImg, numFilters, bins(2)
    %plot(imgSpectralHists(currIndx,:));
```

```
    parfor_progress;
end
parfor_progress(0);
toc;
```

```
Warning: Function externalLib/vlfeat/toolbox/noprefix/det.m has the same n
a MATLAB builtin. We suggest you rename the function to avoid a potential
conflict.
Warning: Function externalLib/vlfeat/toolbox/noprefix/det.m has the same n
a MATLAB builtin. We suggest you rename the function to avoid a potential
conflict.
Reading images and computing the histograms....
100%[===================================================]
Elapsed time is 26.167586 seconds.
```

Average precision and rank for Simplicity method for each class

```
simplicityPR=[  0.47477 178.3529;
                0.32446 242.0187;
                0.33027 261.6305;
                0.36296 260.7511;
                0.98117 49.3074;
                0.39964 197.1079;
                0.40218 298.6917;
                0.71858 91.5890;
                0.34188 230.2441;
                0.33971 271.2211 ];
```

Histogram distances for each method

```
for problem=1:3

    switch problem
        case 1 % Color histogram
            hists = imgHists;
            display('Calculating histogram distances');
            dists = computeHistDist(hists,totalImages);
        case 2 % Spectral histogram
            hists = imgSpectralHists;
            display('Calculating histogram distances');
            dists = computeHistDist(hists,totalImages);
        case 3 % Sift features
            dists=sift(totalImages);
    end
    [Y, ind] = sort(-dists);
```

```
Calculating histogram distances
100%[===================================================]

Calculating histogram distances
```

```
100%[==================================================]

Elapsed time is 7.903175 seconds.
Computing our sift score
```

Precision recall[query image, all images, PR]

```
%every image is queried against all totalImages images
tic;
display('Computing precision recall....');
precision_recall = computePrecRecall(totalImages, ind);
toc;
```

```
Computing precision recall....
100%[==================================================]
Elapsed time is 1.299059 seconds.

Computing precision recall....
100%[==================================================]
Elapsed time is 1.126430 seconds.

Computing precision recall....
100%[==================================================]
Elapsed time is 1.425763 seconds.
```

Average precision and rank

```
display('Computing average precision and rank....');
switch problem
    case 1
        imgHistsPR=computeAvgPR(totalImages, ind, precision_recall);
    case 2
        imgSpectralHistsPR=computeAvgPR(totalImages, ind, precision_recall);
    case 3
        siftPR=computeAvgPR(totalImages, ind, precision_recall);
end
```

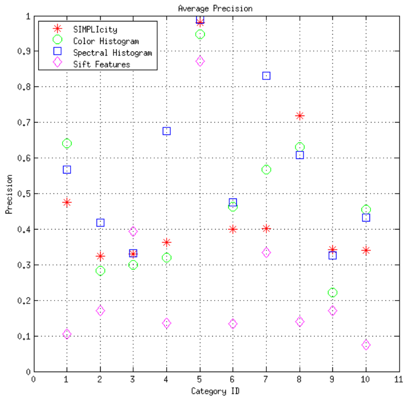
```
Computing average precision and rank....
100%[==================================================]

Computing average precision and rank....
100%[==================================================]

Computing average precision and rank....
100%[==================================================]
```

```
end
```

Plot average precision

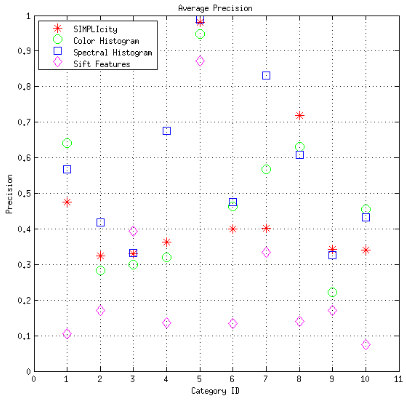
```
figure('Position',[100,100,1500,600])
```

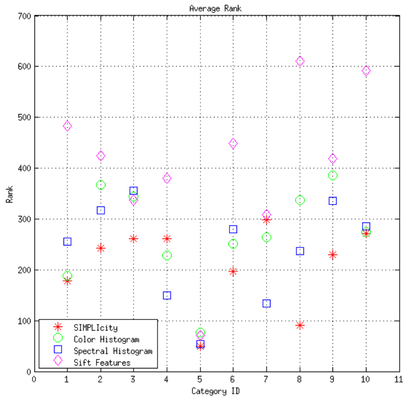
```
subplot(1,2,1);
%Simplicity results
plot(1:10,simplicityPR(:,1),'r*','MarkerSize',10);
hold on
%Our results
plot(1:10,imgHistsPR(:,1),'go','MarkerSize',10);
plot(1:10,imgSpectralHistsPR(:,1),'bs','MarkerSize',10);
plot(1:10,siftPR(:,1),'md','MarkerSize',10);
hold off
title('Average Precision')
xlabel('Category ID');
ylabel('Precision');
legend('SIMPLIcity','Color Histogram','Spectral Histogram','Sift Features');
legend('Location','northwest');
xlim([0,11]);
grid on;
```



Plot average rank

```
subplot(1,2,2);
%Simplicity results
plot(1:10,simplicityPR(:,2),'r*','MarkerSize',10);
hold on
%Our results
plot(1:10,imgHistsPR(:,2),'go','MarkerSize',10);
plot(1:10,imgSpectralHistsPR(:,2),'bs','MarkerSize',10);
plot(1:10,siftPR(:,2),'md','MarkerSize',10);
hold off
title('Average Rank','FontSize',20)
xlabel('Category ID');
ylabel('Rank');
legend('SIMPLIcity','Color Histogram','Spectral Histogram','Sift Features');
legend('Location','southwest');
xlim([0,11]);
grid on;
```





Published with MATLAB® 7.14

Table of Contents


```
function siftScore = sift(totalImages)
```

Calculate descriptors for images

```
tic;
if exist('Variables/descriptors.mat', 'file')
    load('descriptors.mat','descriptors');
else
    display('Detecting SIFT features and descriptors...');
    parfor_progress(totalImages);
    for i=1:totalImages
        [fa, descriptors{i}] = vl_sift(single(rgb2gray(uint8(squeeze(origImages(i,
        parfor_progress;
    end
    save('Variables/descriptors.mat','descriptors');
end
```

Compare descriptors to determine scores

```
if exist('Variables/scores.mat', 'file')
    load('scores.mat','score');
else
    display('Computing Scores...');
    for i=1:totalImages
        tic;
        parfor j=1:totalImages
            [matches, scores{i,j}] = vl_ubcmatch(descriptors{i}, descriptors{j});
        end
        fprintf('%d of 1000: %c',i);
        toc;
    end
    save('Variables/scores.mat','scores');
end
scores = score;
clear score;
toc;
```

Obtaining finalScore

```
% Obtaining max distance
maxDist = 0;
for i=1:totalImages
```

```
    for j=1:totalImages
        currMax = max(scores{i,j});
        if(currMax > maxDist)
            maxDist = currMax;
        end
    end
end

display('Computing our sift score');
siftScore = zeros(totalImages,totalImages);
parfor i=1:totalImages
    for j=1:totalImages
        % Utilize Cantor pairing to ensure one to one mapping for
        % number of matched descriptors with their average
        % distance(Euclidean)
        k1=round(mean(scores{i,j})/maxDist);
        k2=length(scores{i,j});
        siftScore(i,j) = (0.5*(k1+k2)*(k1+k2+1))+k2;
    end
end

end
```

Published with MATLAB® 7.14

```
% The parameter option indicates which filters are we using.
function [filteredImg numFilters] = filterImages(images, option, totalImages)
    switch option
        case 1 %No filters (just the intensity filter)
            filteredImg = images;
            numFilters = 1;
        case 2 % Using just GaussMask
            kernelSize = 5;
            sigma = 0.7;
            filteredImg = zeros(size(images));
            numFilters = 1;
            gaussMask = oz_gaussMask(kernelSize, sigma);
            parfor_progress(length(images));
            for i=1:length(images)
                filteredImg(i,:,:,1) = conv2(squeeze(images(i,:,:,1)),gaussMask, '
                filteredImg(i,:,:,2) = conv2(squeeze(images(i,:,:,2)),gaussMask, '
                filteredImg(i,:,:,3) = conv2(squeeze(images(i,:,:,3)),gaussMask, '
                parfor_progress;
            end
            parfor_progress(0);
            % If you want to visualize any (i) of the filtered images use:
            %imshow(uint8(squeeze(filteredImg(i,:,:,3)))); % 3 is the band of the
        case 3 % Using the 4 filters from the paper
            dimsImages = size(images);
            numFilters = 7;

            if(totalImages > 1) % We have more than one image
                filteredImg = zeros(dimsImages(1)*numFilters, dimsImages(2), dimsI
            else
                filteredImg = zeros(numFilters, dimsImages(1), dimsImages(2), dims
            end

            kernelSize = 5;
            sigma = 0.7;

            theta = 0;
            gabor = gaborFilter(kernelSize, sigma, theta);

            dx = [0 -1 1]; % Range [-256 256]
            dy = [0 -1 1]'; % Range [-256 256]
            dxx = [-1 2 -1];% Range [-512 512]
            dyy = [-1 2 -1]';% Range [-512 512]
            maskLoG = maskLoGFunc(kernelSize, sigma);% Range ?
            gaussMask = oz_gaussMask(kernelSize, sigma);% Range [0 256]

            if(totalImages > 1) % We have more than one image
                parfor_progress(totalImages);
                for i=0:totalImages-1
                    % First filter is the intensity filter
                    filteredImg(i*numFilters+1,:,:,1) = images(i+1,:,:,1);
                    filteredImg(i*numFilters+1,:,:,2) = images(i+1,:,:,2);
                    filteredImg(i*numFilters+1,:,:,3) = images(i+1,:,:,3);
```

```
            % Second filter is dx
            filteredImg(i*numFilters+2,:,:,1) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+2,:,:,2) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+2,:,:,3) = conv2(squeeze(images(i+1,:
            % Third filter is dy
            filteredImg(i*numFilters+3,:,:,1) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+3,:,:,2) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+3,:,:,3) = conv2(squeeze(images(i+1,:
            % Fordth filter is dxx
            filteredImg(i*numFilters+4,:,:,1) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+4,:,:,2) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+4,:,:,3) = conv2(squeeze(images(i+1,:
            % Fifth filter is dxx
            filteredImg(i*numFilters+5,:,:,1) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+5,:,:,2) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+5,:,:,3) = conv2(squeeze(images(i+1,:
            % Sixth filter is LoG
            filteredImg(i*numFilters+6,:,:,1) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+6,:,:,2) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+6,:,:,3) = conv2(squeeze(images(i+1,:
            % Seventh filter is Gauss
            filteredImg(i*numFilters+7,:,:,1) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+7,:,:,2) = conv2(squeeze(images(i+1,:
            filteredImg(i*numFilters+7,:,:,3) = conv2(squeeze(images(i+1,:
            parfor_progress;
        end
        parfor_progress(0);
    else
        % First filter is the intensity filter
        filteredImg(1,:,:,1) = images(:,:,1);
        filteredImg(1,:,:,2) = images(:,:,2);
        filteredImg(1,:,:,3) = images(:,:,3);
        % Second filter is dx
        filteredImg(2,:,:,1) = conv2(images(:,:,1),dx, 'same');
        filteredImg(2,:,:,2) = conv2(images(:,:,2),dx, 'same');
        filteredImg(2,:,:,3) = conv2(images(:,:,3),dx, 'same');
        % Third filter is dy
        filteredImg(3,:,:,1) = conv2(images(:,:,1),dy, 'same');
        filteredImg(3,:,:,2) = conv2(images(:,:,2),dy, 'same');
        filteredImg(3,:,:,3) = conv2(images(:,:,3),dy, 'same');
        % Fordth filter is dxx
        filteredImg(4,:,:,1) = conv2(images(:,:,1),dxx, 'same');
        filteredImg(4,:,:,2) = conv2(images(:,:,2),dxx, 'same');
        filteredImg(4,:,:,3) = conv2(images(:,:,3),dxx, 'same');
        %min(min(filteredImg(i*numFilters+4,:,:,3)))
        %max(max(filteredImg(i*numFilters+4,:,:,3)))
        % Fifth filter is dxx
        filteredImg(5,:,:,1) = conv2(images(:,:,1),dyy, 'same');
        filteredImg(5,:,:,2) = conv2(images(:,:,2),dyy, 'same');
        filteredImg(5,:,:,3) = conv2(images(:,:,3),dyy, 'same');
        % Sixth filter is LoG
        filteredImg(6,:,:,1) = conv2(images(:,:,1),maskLoG, 'same');
        filteredImg(6,:,:,2) = conv2(images(:,:,2),maskLoG, 'same');
        filteredImg(6,:,:,3) = conv2(images(:,:,3),maskLoG, 'same');
```

```
        % Seventh filter is Gauss
        filteredImg(7,:,:,1) = conv2(images(:,:,1),gaussMask, 'same');
        filteredImg(7,:,:,2) = conv2(images(:,:,2),gaussMask, 'same');
        filteredImg(7,:,:,3) = conv2(images(:,:,3),gaussMask, 'same');
    end

case 4 % Intensity and Gauss
    dimsImages = size(images);
    numFilters = 2;

    filteredImg = zeros(dimsImages(1)*numFilters, dimsImages(2), dimsImage
    kernelSize = 5;
    sigma = 0.7;

    gaussMask = oz_gaussMask(kernelSize, sigma);% Range [0 256]

    % TODO missing Gabor filter
    parfor_progress(length(images));
    for i=0:length(images)-1
        % First filter is the intensity filter
        filteredImg(i*numFilters+1,:,:,1) = images(i+1,:,:,1);
        filteredImg(i*numFilters+1,:,:,2) = images(i+1,:,:,2);
        filteredImg(i*numFilters+1,:,:,3) = images(i+1,:,:,3);
        %min(min(filteredImg(i*numFilters+1,:,:,3)))
        %max(max(filteredImg(i*numFilters+1,:,:,3)))
        % Second filter is dx
        %filteredImg(i*numFilters+2,:,:,1) = images(i+1,:,:,1);
        %filteredImg(i*numFilters+2,:,:,2) = images(i+1,:,:,2);
        %filteredImg(i*numFilters+2,:,:,3) = images(i+1,:,:,3);
        % Second filter is dx
        filteredImg(i*numFilters+2,:,:,1) = conv2(squeeze(images(i+1,:,:,1
        filteredImg(i*numFilters+2,:,:,2) = conv2(squeeze(images(i+1,:,:,2
        filteredImg(i*numFilters+2,:,:,3) = conv2(squeeze(images(i+1,:,:,3
        %min(min(filteredImg(i*numFilters+4,:,:,3)))
        %max(max(filteredImg(i*numFilters+4,:,:,3)))
        parfor_progress;
    end
    parfor_progress(0);

    % If you want to visualize any (i) of the filtered images use:
    %imshow(uint8(squeeze(filteredImg(i,:,:,3)))); % 3 is the band of the
end
```

Published with MATLAB® 7.14

```
function hists = computeHist(totalImages, images, numFilters,bins)

    hists=zeros([totalImages,bins*3*numFilters]);%One filter for each band

    if(totalImages>1 || numFilters>1) % We have more than one image
        totRows = size(images(1,:,:,:),2);
        totCols = size(images(1,:,:,:),3);
    else
        totRows = size(images,1);
        totCols = size(images,2);
    end

    minRange = 0;
    maxRange = 256;
    histRange = [minRange:(maxRange-minRange)/(bins-1):maxRange];
    tempSpectralHist = zeros(3*bins*numFilters,1);
    for i=0:totalImages-1
        for j=0:numFilters-1
            % Modifying the range depending on the filter used (it didn't help)
            switch j
                case 0
                    minRange = 0;
                    maxRange = 256;
                case 1
                    minRange = -256;
                    maxRange = 256;
                case 2
                    minRange = -256;
                    maxRange = 256;
                case 3
                    minRange = -512;
                    maxRange = 512;
                case 4
                    minRange = -512;
                    maxRange = 512;
                case 5
                    minRange = 0;
                    maxRange = 256;
                case 6
                    minRange = 0;
                    maxRange = 256;
                case 1
            end

            histRange = [minRange:(maxRange-minRange)/(bins-1):maxRange];

            if(totalImages>1 || numFilters>1) % We have more than one image
                im = images(i*numFilters+1+j,:,:,:);
                im = squeeze(im);%Remove the first 'simgle' dimension
            else
                im = images;
            end
```

```
        % ---- Using bins ---------
        %tempSpectralHist(j*bins*3+1:j*bins*3+bins) = hist(reshape(im(:,:,1),[
        %tempSpectralHist(j*bins*3+bins+1:j*bins*3+2*bins) = hist(reshape(im(:
        %tempSpectralHist(j*bins*3+2*bins+1:j*bins*3+3*bins) = hist(reshape(im

        % ---- Using ranges ---------
        tempSpectralHist(j*bins*3+1:j*bins*3+bins) = hist(reshape(im(:,:,1),[1
        tempSpectralHist(j*bins*3+bins+1:j*bins*3+2*bins) = hist(reshape(im(:,
        tempSpectralHist(j*bins*3+2*bins+1:j*bins*3+3*bins) = hist(reshape(im(
    end

    if(totalImages>1) % We have more than one image
        hists(i+1,:)= tempSpectralHist;
    else
        hists= tempSpectralHist;
    end
end
```

Published with MATLAB® 7.14

```
function dists = computeHistDist(hists,totalImages)

    parfor_progress(totalImages);% External library Copyright (c) 2011, Jeremy Sch
    dists=zeros([totalImages,totalImages]);
    parfor i=1:totalImages
        parfor_progress;
        for j=1:totalImages
            dists(i,j)=sum(min(hists(i,:),hists(j,:)));
        end
    end
    parfor_progress(0);
```

Published with MATLAB® 7.14

This function creates a Gaussian mask of nxn

```
function gauss2d = maskLoGFunc(n,sigma)

T = sqrt(2)*sigma;
f = @(x,y,sigma) ( x.^2 + y.^2 - T.^2 ) .* exp( -(x.^2+y.^2)/T^2 );

width = 5;
height = 5;
x = [-width/2:width/n:width/2];
y = [-height/2:height/n:height/2];

[X,Y] = meshgrid(x,y);

gauss2d= f(X,Y,sigma);

%Normalization of the gauss function
total = sum(sum(gauss2d));
gauss2d = gauss2d./total;%It doens't validate that the minimum can be 0

%surf(gauss2d);
```

Published with MATLAB® 7.14

```
function images = readImages(totalImages, imgPath)

    % Read one image and obtain the dimensions
    fname=sprintf('%s/%i.jpg',imgPath,0);
    tempImg = double(imread(fname,'jpg'));
    dim = size(tempImg);
    images = zeros(totalImages, dim(1), dim(2), dim(3));
    parfor_progress(totalImages);
    for i=1:totalImages
        fname=sprintf('%s/%i.jpg',imgPath,i-1);
        clear tempImg;
        tempImg = imread(fname,'jpg');
        if( size(tempImg,1) ~= dim(1) ) %The image is rotated
            images(i,:,:,1)= tempImg(:,:,1)';
            images(i,:,:,2)= tempImg(:,:,2)';
            images(i,:,:,3)= tempImg(:,:,3)';
        else
            images(i,:,:,:)= tempImg;
        end
        parfor_progress;
    end
    parfor_progress(0);
```

Published with MATLAB® 7.14