TERM PROJECT

SIFT ALGORITHM

January 5,2019

Barış Sevilmiş, 20945

Yiğit Aras Tunalı, 17519

EE417-Computer Vision

List of Figures

Contents

.1 INTRODUCTION

The aim of this project is to study and implement the SIFT algorithm developed by David G. Lowe. The algorithm is mainly studied from the paper published by David G. Lowe in 2004 (2) and necessary citations will be made and the points taken from the paper will all be shown properly throughout this report. Additionally, any extra resources used and studied will also be cited and shown properly.

SIFT algorithm aims to find image features which are invariant to scaling, rotation, change in illumination and 3D camera viewpoints and one other aim is to extract these features efficiently. Their good localization in the spatial and frequency domains makes them have reduced probability of disruption by occlusion, clutter or noise (2). These features then can be matched and used for solving the 3D structure, stereo correspondence, and motion tracking problems.

The SIFT algorithm tries to minimize the use of expensive extraction operations on locations that pass initial filtering operations. The main steps of computation that the SIFT algorithm follows (2) are listed below ;

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

After finding and storing the extracted features from multiple, in this case 2, different photographs of a scene taken from different perspectives, we then match these found features of the two images and try to measure the performance of our implementation by the analysis of this matching and the quality of features extracted manually. Since the keypoint descriptors are considered to be highly distinctive, we expect features to find correct matches with a good probability. For the matching functionality, we used a built-in function of MATLAB i.e., matchFeatures function. The resulting images and calculations will also be provided in the sections of each step with their corresponding code, implemented by us.

.2 Steps Of The Algorithm

In this section, the steps of the SIFT algorithm will be discussed further with its mathematical formulas and steps and the corresponding code blocks for these steps will be provided. Below is the initialization phase of the algorithm;

```
%% Initialize phase
row = zeros(4,1) ;
col = zeros(4,1);
[row(1), col(1), ch] = size(img);
if(ch == 3)
    new_img = rgb2gray(img);
else
    new_img = img;
end
new_img = double(new_img)./255;
scale_space = cell(5,4);
dog_space = cell(4,4);
sigma_arr = zeros(5,4);
% Below values are taken directly from Lowe (1)
sigma = 1.6;
k = sqrt(2);
```

We first allocate necessary matrices, turn our image into grayscale if it is not, then re-scaling the pixel values to [0 1] by dividing each pixel value to 255, creating an empty scale and DOG (difference of gaussians) space and a sigma array for further use, whose usage will be explained in their respective sections. As can be seen from the above code block the sigma value is fixed as 1.6 and k value is fixed as $\sqrt{2}$, these values are taken from Lowe (2). These values are found after testing and sampling for both in their respective tests.

.2.1 Scale-space Extrema Detection

The detection of keypoints is done by using a cascade filtering approach then the candidate locations are examined further by identifying them with efficent algorithms. The first step is to detect locations that are invariant to scale change, which will be done by searching through all possible scales using a continuous function of scale known as scale space (2).

The function used for building the scale space will be $L(x, y, \sigma)$ where it is defined as;

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

According to Lowe (2) and the citations in his paper, this method is proven to be the only possible scale-space kernel. We build the scale space with the following code block;

```
new_img = imresize(new_img, 2 , 'bilinear');
for ii=1:size(scale_space,2)
    row(ii) = size(new_img,1);
    col(ii) = size(new_img,2);
    counter = 2*ii - 2;
    for jj=1:size(scale_space,1)
        scl = sigma * (k^counter);
        scale_space{jj,ii} = imgaussfilt(new_img, scl);
        sigma_arr(jj,ii) = scl;
        counter = counter + 1;
    end
    new_img = imresize(new_img, 0.5, 'bilinear');
end
```

We re-size the image to twice its size with bilinear interpolation and start building our scale-space. The counter is used for determining the factor that we will multiply with our sigma value which we will use for our Gaussian Filter. After the end of each nested for-loop our image is re-sized down to half it's size and another octave is built using the same routine. The first sigma value of each octave is the k^2 multiplied one of the latter octave, where k is the scale factor that we multiply with the sigma. The scale-space created from our example pictures are as follows(We couldn't show smaller images as smaller, so they all look the same size but actually as you go right and down images get smaller and blurrier);

Figure 1: Scale-space Example Images 1

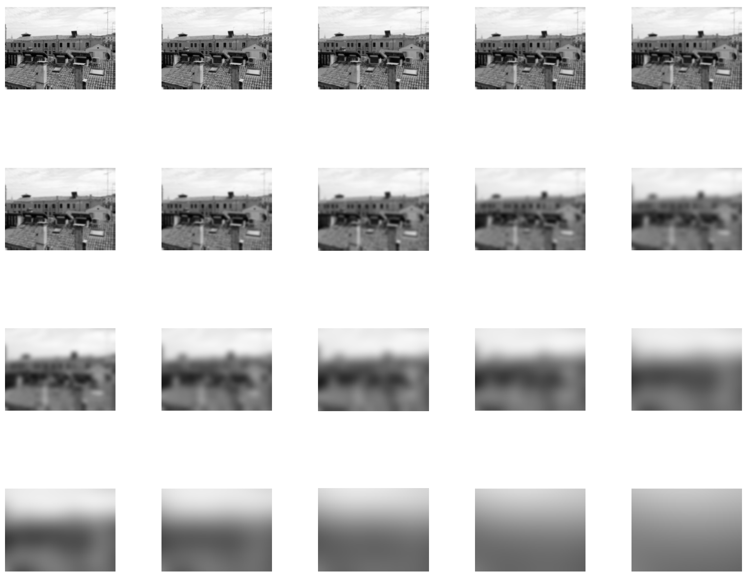


Figure 2: Scale-space Example Images 2

The next step in the algorithm is to use the scale-space extrema in difference-of-Gaussian function convolved with the image,which is $D = (x, y, \sigma)$ and will be computed from the difference of scales that are separated by a constant multiplicative factor k, as shown in the last code-block above. Mathematical formulation for this operation is as follows;

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

$$= L(x, y, k\sigma) - L(x, y, \sigma)$$

The figure explaining the structure of DOG-Space is as follows;

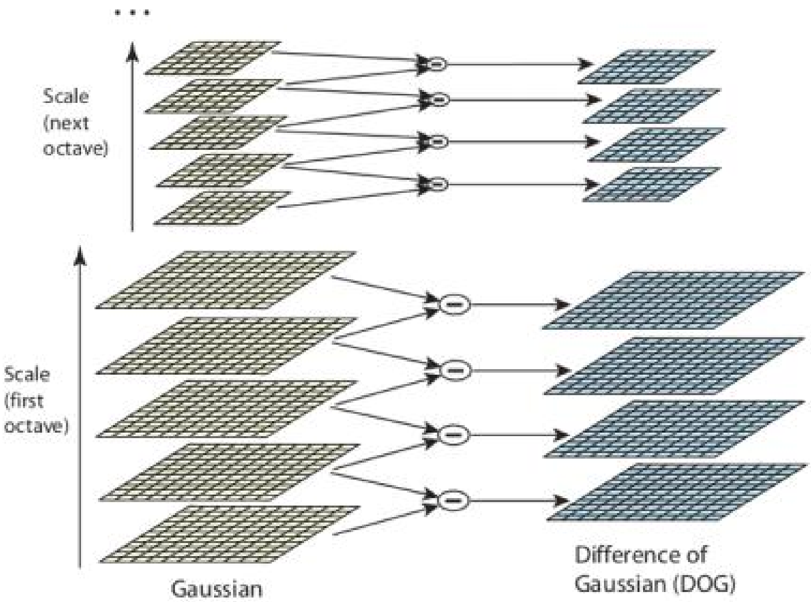


Figure 3: Structure of DOG-Space

Our code for this task is as follows;

```
%% Create DoG space(Difference of Gaussians)
for kk=1:size(dog_space,2)
    for mm=1:size(dog_space,1)
    dog_space{mm,kk}=imsubtract(scale_space{mm+1,kk},scale_space{mm,kk});
    end
end
```

It is shown in Lowe 2004 (2) that difference-of-Gaussian function has scales differing by a constant factor and it already includes the σ^2 scale normalization for scale in-varint Laplacian. Also the below formulation is shown to be true and with this assumption we will proceed to maxima and minima detection;

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$$

The $(k-1)$ value in the equation is a constant across the over all scales so it doesn't affect extrema location.

.2.1.1 Local Extrema Detection

For the detection of local maxima and minima of $D(x, y, \sigma)$, we will compare each sample point to it's eight adjacent neighbours in the current image and nine neighbours in the scale above and below. The following figure demonstrates the comparison schema, where X is the pixel we want to compare and green pixels are the all other ones we do the comparison with;

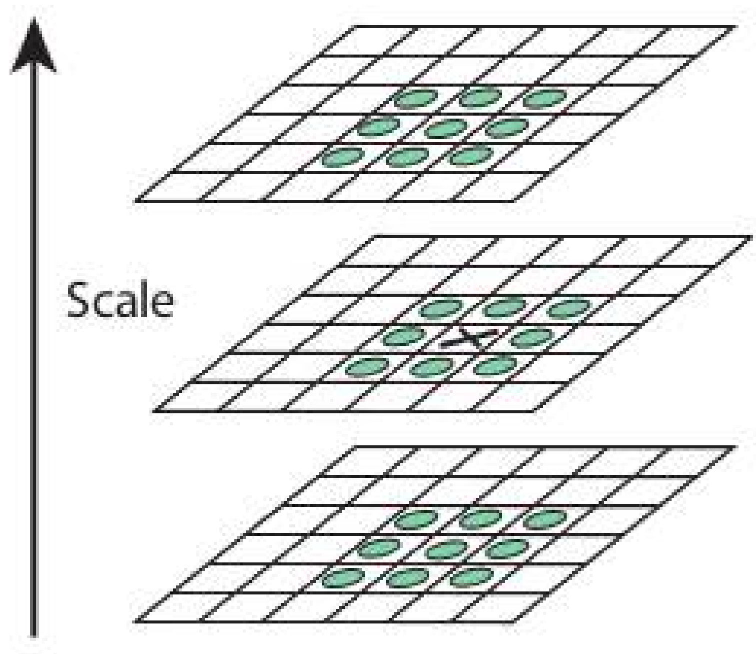
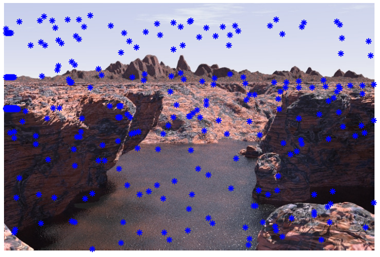


Figure 4: The Comparison Schema

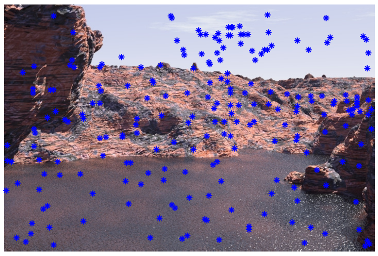
Our code for extrema detection, in the neighbourhood of the pixel is as follows;

```
% First two index for image index, last two for min and max coord.
coord = [];k = 1;
for aa = 1:size(dog_space,2)
    main2 = dog_space{2, aa};main3 = dog_space{3, aa};
    neighof2_1 = dog_space{1, aa};neighof2_2 = dog_space{3, aa};
    neighof3_1 = dog_space{2, aa};neighof3_2 = dog_space{4, aa};
for bb = k + 1: 1: row(aa) - k -1  %Change step size
    for cc = k + 1: 1: col(aa) - k - 1
        %Case 1
        main2_window = main2(bb - k: bb + k, cc - k: cc + k);
        neighof2_1window = neighof2_1(bb - k: bb + k, cc - k: cc + k);
        neighof2_2window = neighof2_2(bb - k: bb + k, cc - k: cc + k);
        if (main2(bb, cc) == max(main2_window(:))) && (main2(bb, cc) > max(neighof2_1window(:)))...
                && (main2(bb, cc) > max(neighof2_2window(:)))
            coord = [coord; 2 aa bb cc sigma_arr(2,aa)];
        elseif (main2(bb, cc) == min(main2_window(:))) && (main2(bb, cc) < min(neighof2_1window(:)))...
                    && (main2(bb, cc) < min(neighof2_2window(:)))
            coord = [coord; 2 aa bb cc sigma_arr(2,aa)];
        end
        %Case 2
        main3_window = main3(bb - k: bb + k, cc - k: cc + k);
        neighof3_1window = neighof3_1(bb - k: bb + k, cc - k: cc + k);
        neighof3_2window = neighof3_2(bb - k: bb + k, cc - k: cc + k);
            if (main3(bb, cc) == max(main3_window(:))) && (main3(bb, cc) > max(neighof3_1window(:)))...
                    && (main3(bb, cc) > max(neighof3_2window(:)))
                coord = [coord; 3 aa bb cc sigma_arr(3,aa)];
            elseif (main3(bb, cc) == min(main3_window(:))) && (main3(bb, cc) < min(neighof3_1window(:)))...
                    && (main3(bb, cc) < min(neighof3_2window(:)))
                coord = [coord; 3 aa bb cc sigma_arr(3,aa)];
end end end end
```

A point is selected only if it is larger than or smaller than all the compared neighbours, as done in the code block above. Below is the non-thresholded found points on one particular example with exact number of keypoints detected;



(a) A part of the Image: 25279 keypoints



(b) B part of the Image: 18098 keypoints

Figure 5: Non-Thresholded Points Found from Both Images

To be clear, keypoints from each DoG image are considered as different keypoints, although some of them may be referring to same keypoints. However, this redundancy of keypoints are solved in following thresholding steps.

.2.2 Accurate Keypoint Localization

After finding candidate pixels, we now will try to perform a fit to nearby data for location, scale, and the ratio of principal curvature. With this step, the points that have low contrast, sensitive to noise, and ones that are poorly localized along an edge will be rejected. The approach we will use here is the one discussed in Lowe (2), where the Taylor expansion up to quadratic term of the scale-space function $D(x, y, \sigma)$ which is shifted to the origin is used. The derivatives and D are evaluated at sample point and there is an offset from this point, namely $x = (x, y, \sigma)^T$ from the said point. The $D(x)$ is as follows:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x$$

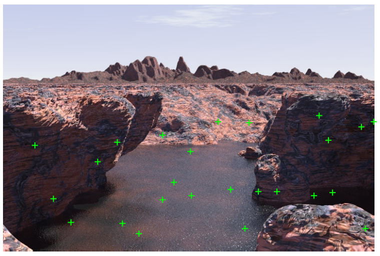
The location of the extremum, $\hat{x}$, will be found by taking the derivative of the upper formula with respect to x and setting it to zero, thus reaching:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

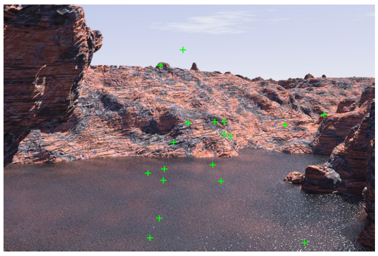
Hessian and the derivative of D will be approximated by the differences of neighboring points, obtaining a 3x3 linear system. This 3x3 linear system can be formulated as follows:

$$\frac{\partial^2 D}{\partial x^2} = \begin{bmatrix} D_{xx} & D_{xy} & D_{xs} \\ D_{xy} & D_{yy} & D_{ys} \\ D_{xs} & D_{ys} & D_{ss} \end{bmatrix}$$

Calculation of the each of these elements are depicted in Appendix under Removing Non-Maximal/Minimal Outliers part. Another important part is that the thresholds that we will be using at this point. First of all, we have a threshold for $|D(\hat{x}|$, if its value is smaller than 0.03 we discard it immediately. The second threshold we have is that if any point's value we find with the formula above, that we obtain by taking the derivative, is smaller than 0.5. The reason for this is that if a point has a value bigger or equal to 0.5, it will lie closer to a different sample point. Both these values are taken from Lower (2). The code for this task is a bit too long and it wouldn't fit here, it will be added with all the rest of the code to the appendix. After this first round of elimination of points, the results we have are as follows again with the number of remaining keypoints;



(a) A part of the Image: 1179 keypoints



(b) B part of the Image: 1567 keypoints

Figure 6: Result after the first thresholding

It is clearly shown that keypoint amount reduces enormously after keypoint localization step. Amount of difference before and after thresholding are demonstrated in Figure 5 and Figure 6.

.2.2.1 Eliminating Edge Responses

It is stated in Lowe (2) that the elimination step we have done in the prior step alone is not enough for stability. The DoG function has a strong response along edges even if it is poorly determined, and remains unstable against noise. For this second step of thresholding elimination we will have to find the poorly defined peaks in the DOG function. These places will have a large principal curvature across the edges and a small one in the perpendicular direction. For the curvature we will use the 2x2 Hessian matrix as defined below;

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

It is stated that the eigenvalues of this H, will be proportionate to the principal curvatures of D. Thus not needing to explicitly calculate the eigenvalues. If α is the eigenvalue with largest magnitude and β be the smallest magnitude, their product and the sum can be calculated using the determinant of H and trace of H as follows (exactly as we have done in the lectures and labs);

$$Tr(\boldsymbol{H}) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(\boldsymbol{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

The points with negative determinants are discarded first. The proposed way of checking the ratio of principal curvature is to select a threshold, in this case is $r = 10$, and compute efficently;

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

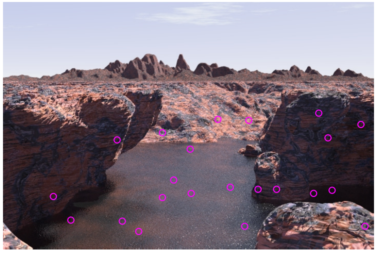
Following r value is chosen according to (2). Using the above formula we can show that checking the below inequality is enough;

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \frac{(r+1)^2}{r}$$

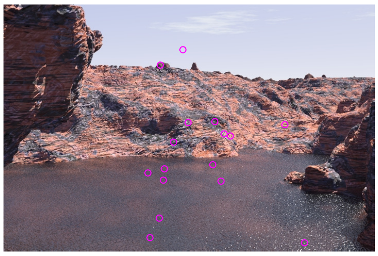
With the above formulation the points that have a ratio between the principal curvatures greater than 10 are eliminated. The code for this task is as follows (as usual the full code will be included in the appendix as a whole);

```matlab
%% Eliminate  Edge Responses
coord_final = [];
for ll=1:size(coord3,1)
    D = dog_space{coord3(ll,1), coord3(ll,2)};
    Dxx_new = D(coord3(ll, 3), coord3(ll, 4) + 1)...
        + D(coord3(ll, 3), coord3(ll, 4) - 1)...
        - (2*D(coord3(ll, 3), coord2(ll, 4)));
    Dxy_new = (D(coord3(ll, 3) + 1, coord3(ll, 4) + 1)...
        - D(coord3(ll, 3) - 1, coord3(ll, 4) + 1)...
        -D(coord3(ll, 3) + 1, coord3(ll, 4) - 1)...
        + D(coord3(ll, 3) - 1, coord3(ll, 4) - 1))/4;
    Dyy_new = D(coord3(ll, 3) + 1, coord3(ll, 4))...
        + D(coord3(ll, 3) - 1, coord3(ll, 4))...
        - (2*D(coord3(ll, 3), coord3(ll, 4)));
    H2 = [Dxx_new Dxy_new; Dxy_new Dyy_new];     %Hessian 2x2
    Tr_H2 = H2(1,1) + H2(2,2);
    Det_H2 = (H2(1,1)*H2(2,2)) - (H2(1,2)*H2(2,1));
    %Evaluate Edge Responses
    r = 10;
    Ev_H2 = ((Tr_H2)^2)/Det_H2;
    if Ev_H2  < (((r+1)^2)/r)
        coord_final = [coord_final; coord3(ll, :)];
end end
```

The points that remain after the final thresholding are as follows;



(a) A part of the Image: 1109 keypoints



(b) B part of the Image: 1509 keypoints

Figure 7: Result after the last thresholding

.3 Orientation Assignment

Next step is the so-called "Orientation Assignment". The aim of this step is to assign keypoints some consistent orientation regarding the local properties of the image, thus achieving invariance to rotation. In Lowe (2) it is stated that after experimentation with a number of approaches the method they chose for this step is as follows. The scale of the keypoint is used to select the Gaussian smoothed image, L, to the closest scale and thus computing the operations in a scale-invariant manner. For each $L(x, y)$ at that scale the $m(x, y)$ and $\theta(x, y)$ are pre-computed using the following formulas;

$$m(x,y) = \sqrt{(L(x+1,y)-L(x-1,y))^2 + (L(x,y+1)-L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1)-L(x,y-1))/(L(x,y-1)-L(x-1,y)))$$

Magnitude and Scale space are constructed as follows in code;

```
%% Magnitude and Theta Space
mag_space = cell(5,4);
theta_space = cell(5,4);
offset = 50;
for ii = 1:size(scale_space,2)
    for jj=1:size(scale_space,1)
        mag_space{jj, ii} = zeros(size(scale_space{jj,ii},1),...
            size(scale_space{jj,ii},2));
        theta_space{jj, ii} = zeros(size(scale_space{jj,ii},1),...
            size(scale_space{jj,ii},2));
        I = scale_space{jj,ii};
        for xx=2:size(mag_space{jj, ii},1)-1
            for yy=2:size(mag_space{jj, ii},2)-1
                mag_space{jj, ii}(xx,yy) = ...
                    sqrt(((I(xx + 1, yy)-I(xx -1,yy))^2)...
                    + ((I(xx,yy+1)-I(xx,yy-1))^2));
                theta_space{jj, ii}(xx,yy) = ...
                    atan2d((I(xx,yy+1)-I(xx,yy-1))...
                    , (I(xx+1,yy)-I(xx-1,yy)));
                theta_space{jj, ii}(xx,yy) = ...
                    mod(theta_space{jj, ii}(xx,yy) + 360,360);
            end
        end
        mag_space{jj, ii} = padarray(mag_space{jj, ii},...
            [offset offset],'both');
        theta_space{jj, ii} = padarray(theta_space{jj, ii},...
            [offset offset],'both');
    end
end
```

After setting up the magnitude and theta spaces, we create an orientation histogram from the gradient orientations of sample points in an area around the keypoints. In (3), this process is explained in a detailed manner. Orientation histogram consists of 36 bins ranging the whole 360-degree range of orientations. Each sample is added to the orientation histogram by weighting it by its gradient magnitude and by a Gaussian-weighted circular window with σ that is 1.5 times the scale of the keypoint. Although (3) suggests that 9x9 Gaussian window with 1.5*$*\sigma$ gives accurate results, in our work, we have chosen our Gaussian window size relative to the 1.5$*\sigma$ as in (1). The code for the orientation assignment part is a bit longer and as before it can be found in Appendix A, the code is chopped into blocks with headlines as comments clarifying which phase it is.

Peaks in the histogram are related to the dominant directions of the local gradients. We detect the highest peak in the histogram, followed by any other local peak that is within %80 of the first found peak. They are all used to create new keypoint with that orientation. (There will be about %15 of points that have multiple orientations, but this is said to reinforce the stability of matching, Lowe (2)). Finally, we do linear interpolation, by fitting a parabola, to the 3 histogram values closest to each peak, thus acquiring better accuracy. In other words, interpolation is done by fitting a parabola to the $+/-1$ bins of the dominant peaks.

.4 THE LOCAL IMAGE DESCRIPTOR

For this step, we have tried the procedure in Lowe 2004 (2) but received accurate results up to a certain rotational degree. Unfortunately, SIFT is not totally rotation invariant but is rotation invariant up to a certain rotation. The main idea behind this step is to, for each keypoint, a keypoint descriptor is computed by taking sampling points around the keypoint, weighted by a Gaussian window and accumulating these into orientation histograms which represent the contents over a 4x4 subregion. In order to achieve rotational invariance, scale space images are rotated relative to dominant orientation. Furthermore, keypoint locations are computed within the rotated scale space image by using a rotation matrix. By using the rotation matrix, as we have seen in lectures, keypoint location is detected within the rotated image. As in (2) & (1), $16x16$ window is built around rotated keypoint, which is divided into 4x4 subregions. A $16x16$ gaussian weighted window is used to balance the magnitudes, such that distant pixels from keypoint do not have the same level of effect as near pixels. These subregions are again put into histogram bins as large as their gaussian weighted magnitude according to their orientations. However, 8 histogram bins each covering 45 degrees, from 0 degree to 360 degrees, are created for every $4x4$ window. There are 16 such subregions and for every subregion, 8 bins are formed, therefore feature vector for a keypoint is of size $16 * 8$, namely of size 128. We placed that part in our code, commented out, and will compare the results we received from another source's implementation;

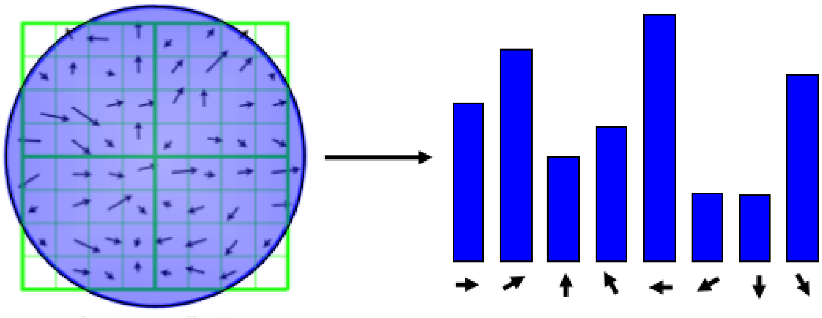


Figure 8: Sift Orientation Representation

The other resource that was used for this step was (1). Our initial approach was based on this method, such that it was much easier to implement. Here it is proposed, that the rotation of each keypoint to be subtracted from each orientation, thus making gradient orientations relative to the keypoint's orientation. In other words, no rotations to images or keypoints are computed, but the dominant orientation of keypoint is subtracted from each pixel orientation following with a $mod(360)$ as in our first explained approach. To be more precise, these subtracted rotations may fall in negative orientation range, therefore as done previously in the creation of orientation space, we computed $mod(360)$ of each orientation in terms of mapping orientation into a range of $[0, 360]$. Although this approach seems much more inaccurate, results were as good as our first approach. More matches are detected, and most of these matches are accurate. Moreover, in some cases as Landscape images, this approach proved to be more accurate. The code blocks for this task are added in Appendix A part.

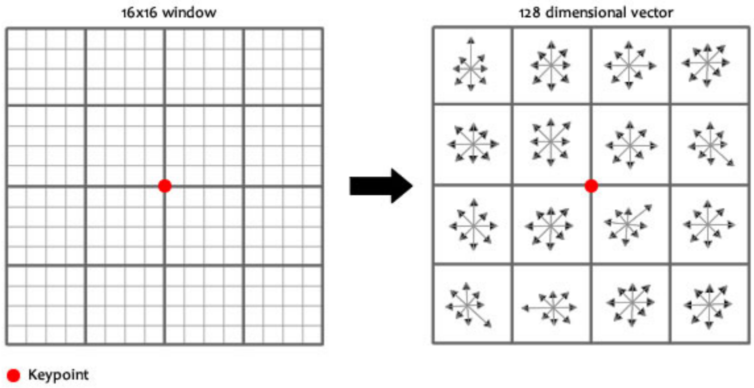


Figure 9: Creating of Unique Keypoint Fingerprint

The example of how the surrounding sample points of a keypoint is used for creating a descriptor is also shown above, image taken from (1)

.5 Feature Matching

Finally, with the use of MATLAB's built-in matching function, we match the features we accumulated in two different feature vectors with each other and display our results. By default, matches are done using Sum Of Squared Distances(SSD). The threshold is chosen as 0.6 and if the threshold is decreased, then more matches are found. However, these matches are less accurate. Therefore, the threshold is chosen as 0.6. Lastly, only unique matches are drawn to reduce the visual complexity of matches and additionally, non-unique matches do not provide accurate results. Lastly, images for matching and detection were mostly taken from (4) such as the roof images. Related implementation is demonstrated in Appendix B. Our results are as follows;

Figure 10: Results using Lowe's Approach

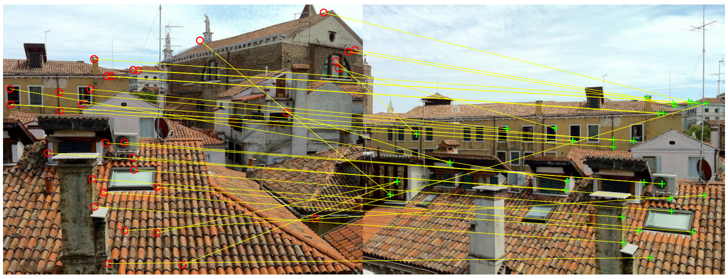


Figure 11: Using (1) Approach

As can be seen from the two examples, the number of matches is increased with the second approach. Another result on a different image is as follows;

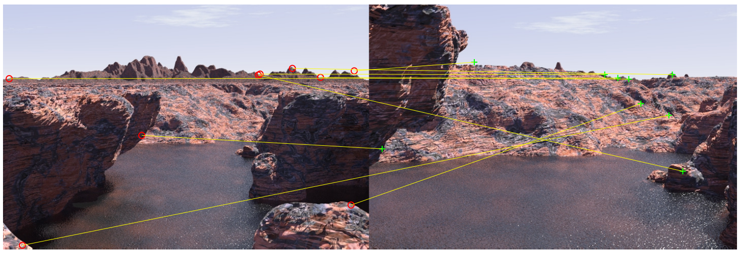


Figure 12: Using Lowe's Approach

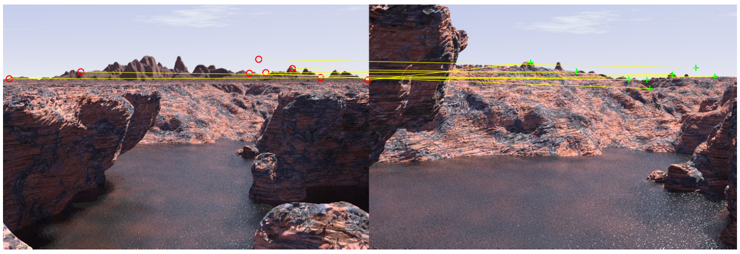


Figure 13: Using Approach in (1)

Here we can see that the second approach outperforms the first one again.

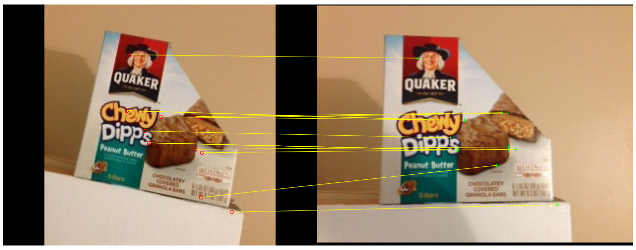


Figure 14: Another Result on a Simpler Image, (1)

As can be seen from the above image, the matched features are pretty solid.

.6 Appendix A

```
% clear all; close all;clc;
% img = imread('dataset/landscape-b.jpg');
function [feature_vec, validpoints] = siftfeature(img)
%% Initialize phase
row = zeros(4,1) ;
col = zeros(4,1);
[row(1), col(1), ch] = size(img);

if(ch == 3)
    new_img = rgb2gray(img);
else
    new_img = img;
end

new_img = double(new_img)./255;
scale_space = cell(5,4);
dog_space = cell(4,4);
sigma_arr = zeros(5,4);

sigma = 1.6;
k = sqrt(2);

%% Create scale space with sigma = 1.6 and k = sqrt(2)
new_img = imresize(new_img, 2, 'bilinear');
for ii=1:size(scale_space,2)
    row(ii) = size(new_img,1);
    col(ii) = size(new_img,2);
    counter = 2*ii - 2;
    for jj=1:size(scale_space,1)

        scl = sigma * (k^counter);
        scale_space{jj,ii} = imgaussfilt(new_img, scl);
        sigma_arr(jj,ii) = scl;
```

```
            counter = counter + 1;
        end
        new_img = imresize(new_img, 0.5, 'bilinear');
end
%figure(1);title(scale_space);
%% Create DoG space(Difference of Gaussians)
for kk=1:size(dog_space,2)
    for mm=1:size(dog_space,1)
        dog_space{mm,kk} = imsubtract(scale_space{mm+1, kk},scale_space{mm, kk});
    end
end

%% Local Maxima and Minima Detection
% Window size = 3, 3
k = 1;
% First two index for image index, last two for min and max coord.
coord = [];
for aa = 1:size(dog_space,2)
    main2 = dog_space{2, aa};
    main3 = dog_space{3, aa};
    neighof2_1 = dog_space{1, aa};
    neighof2_2 = dog_space{3, aa};
    neighof3_1 = dog_space{2, aa};
    neighof3_2 = dog_space{4, aa};

    %Change step size
    for bb = k + 1: 1: row(aa) - k -1
        for cc = k + 1: 1: col(aa) - k - 1
            %Case 1
            main2_window = main2(bb - k: bb + k, cc - k: cc + k);
            neighof2_1window = neighof2_1(bb - k: bb + k, cc - k: cc + k);
            neighof2_2window = neighof2_2(bb - k: bb + k, cc - k: cc + k);

            if (main2(bb, cc) == max(main2_window(:)))...
                    && (main2(bb, cc) >= max(neighof2_1window(:)))...
```

```
                    && (main2(bb, cc) >= max(neighof2_2window(:)))

                coord = [coord; 2 aa bb cc sigma_arr(2,aa)];

            elseif (main2(bb, cc) == min(main2_window(:)))...
                    && (main2(bb, cc) <= min(neighof2_1window(:)))...
                    && (main2(bb, cc) <= min(neighof2_2window(:)))

                coord = [coord; 2 aa bb cc sigma_arr(2,aa)];
            end

            %Case 2
            main3_window = main3(bb - k: bb + k, cc - k: cc + k);
            neighof3_1window = neighof3_1(bb - k: bb + k, cc - k: cc + k);
            neighof3_2window = neighof3_2(bb - k: bb + k, cc - k: cc + k);


            if (main3(bb, cc) == max(main3_window(:)))...
                    && (main3(bb, cc) >= max(neighof3_1window(:)))...
                    && (main3(bb, cc) >= max(neighof3_2window(:)))

                coord = [coord; 3 aa bb cc sigma_arr(3,aa)];

            elseif (main3(bb, cc) == min(main3_window(:)))...
                    && (main3(bb, cc) <= min(neighof3_1window(:)))...
                    && (main3(bb, cc) <= min(neighof3_2window(:)))

                coord = [coord; 3 aa bb cc sigma_arr(3,aa)];
            end

        end
    end
end

%Display non-thresholded corner points
```

```
disp(size(coord,1));
figure;
imshow(img);
hold on;
plot((2.^coord(:,2)-2).*coord(:,4), (2.^coord(:,2)-2).*coord(:,3) , 'b*');

%% Remove non-maximal/minimal outliers
coord2 = [];
% Threshold_value
threshold_1 = 0.03;
for ll=1:size(coord,1)

    % Hesssian Matrix Values
    D = dog_space{coord(ll,1), coord(ll,2)};
    Dx = (D(coord(ll,3), coord(ll,4) + 1)...
        - D(coord(ll,3), coord(ll,4) - 1))/2;
    Dy = (D(coord(ll,3) + 1, coord(ll,4))...
        - D(coord(ll,3) - 1, coord(ll,4)))/2;
    Ds = (dog_space{coord(ll,1) + 1, coord(ll,2)}...
        (coord(ll,3), coord(ll,4))...
        - dog_space{coord(ll,1) - 1, coord(ll,2)}...
        (coord(ll,3), coord(ll,4)))/2;
    Dxx = D(coord(ll, 3), coord(ll, 4) + 1)...
        + D(coord(ll, 3), coord(ll, 4) - 1)...
        - (2*D(coord(ll, 3), coord(ll, 4)));
    Dyy = D(coord(ll, 3) + 1, coord(ll, 4))...
        + D(coord(ll, 3) - 1, coord(ll, 4))...
        - (2*D(coord(ll, 3), coord(ll, 4)));
    Dxy = (D(coord(ll, 3) + 1, coord(ll, 4) + 1)...
        - D(coord(ll, 3) - 1, coord(ll, 4) + 1)...
        -D(coord(ll, 3) + 1, coord(ll, 4) - 1)...
        + D(coord(ll, 3) - 1, coord(ll, 4) - 1))/4;
    Dxs = (dog_space{coord(ll,1) + 1, coord(ll,2)}...
        (coord(ll, 3), coord(ll, 4) + 1)...
        - dog_space{coord(ll,1) + 1, coord(ll,2)}...
```

```
        (coord(ll, 3), coord(ll, 4) - 1)...
        - dog_space{coord(ll,1) - 1, coord(ll,2)}...
        (coord(ll, 3), coord(ll, 4) + 1)...
        + dog_space{coord(ll,1) - 1, coord(ll,2)}...
        (coord(ll, 3), coord(ll, 4) - 1))/4;
    Dys = (dog_space{coord(ll,1) + 1, coord(ll,2)}...
        (coord(ll, 3) + 1, coord(ll, 4))...
        - dog_space{coord(ll,1) + 1, coord(ll,2)}...
        (coord(ll, 3) - 1, coord(ll, 4))...
        - dog_space{coord(ll,1) - 1, coord(ll,2)}...
        (coord(ll, 3) + 1, coord(ll, 4))...
        + dog_space{coord(ll,1) - 1, coord(ll,2)}...
        (coord(ll, 3) - 1, coord(ll, 4)))/4;
    Dss = (dog_space{coord(ll,1) + 1, coord(ll,2)}...
        (coord(ll, 3), coord(ll, 4))...
        + dog_space{coord(ll,1) - 1, coord(ll,2)}...
        (coord(ll, 3), coord(ll, 4)))...
        - (2*D(coord(ll, 3), coord(ll, 4)));

    %Clairaut's Theorem: Dxy == Dyx iff. both are continuous within
    %their definition range
    H1 = [Dxx Dxy Dxs; Dxy Dyy Dys; Dxs Dys Dss];
    %Loc_ext => x,y,s

    D_dif = [Dx; Dy ;Ds];
    loc_ext = (-pinv(H1)) * D_dif;
    Taylor_dog = D(coord(ll, 3), coord(ll, 4)) + ((D_dif')*loc_ext)/2;

    if (abs(Taylor_dog) > threshold_1) && (max(abs(loc_ext)) < 0.5)
        coord2 = [coord2; coord(ll,:) (coord(ll,3)+loc_ext(1))...
            (coord(ll,4)+loc_ext(2)) (coord(ll,5)+loc_ext(3)) Taylor_dog];
    end

end
```

```
%Display first_level-thresholded corner points
disp(size(coord2, 1));
plot((2.^coord2(:,2)-2).*coord2(:,4),(2.^coord2(:,2)-2).*coord2(:,3),'g+');

%% Eliminate Low Contrast Extremum

coord3 = [];
for ll = 1:size(coord2,1)
    D = dog_space{coord2(ll,1), coord2(ll,2)};
    C_dog = 0.015;
    if (abs(coord2(ll, 9)) >= C_dog)...
            && (abs(D(coord2(ll, 3), coord2(ll, 4))) >=  0.8*C_dog)
        coord3 = [coord3; coord2(ll, :)];
    end
end
disp(size(coord3,1));

%% Eliminate  Edge Responses
coord_final = [];
for ll=1:size(coord3,1)

    D = dog_space{coord3(ll,1), coord3(ll,2)};
    Dxx_new = D(coord3(ll, 3), coord3(ll, 4) + 1)...
        + D(coord3(ll, 3), coord3(ll, 4) - 1)...
        - (2*D(coord3(ll, 3), coord2(ll, 4)));
    Dxy_new = (D(coord3(ll, 3) + 1, coord3(ll, 4) + 1)...
        - D(coord3(ll, 3) - 1, coord3(ll, 4) + 1)...
        -D(coord3(ll, 3) + 1, coord3(ll, 4) - 1)...
        + D(coord3(ll, 3) - 1, coord3(ll, 4) - 1))/4;
    Dyy_new = D(coord3(ll, 3) + 1, coord3(ll, 4))...
        + D(coord3(ll, 3) - 1, coord3(ll, 4))...
        - (2*D(coord3(ll, 3), coord3(ll, 4)));

    %Hessian 2x2
    H2 = [Dxx_new Dxy_new; Dxy_new Dyy_new];
```

```
    Tr_H2 = H2(1,1) + H2(2,2);
    Det_H2 = (H2(1,1)*H2(2,2)) - (H2(1,2)*H2(2,1));

    %Evaluate Edge Responses
    r = 10;
    Ev_H2 = ((Tr_H2)^2)/Det_H2;
    if Ev_H2  < (((r+1)^2)/r)
        coord_final = [coord_final; coord3(ll, :)];
    end
end
disp(size(coord_final,1));
plot((2.^coord_final(:,2)-2).*coord_final(:,4),...
    (2.^coord_final(:,2)-2).*coord_final(:,3) , 'mo');
hold off;
%% Magnitude and Theta Space
mag_space = cell(5,4);
theta_space = cell(5,4);
offset = 50;
for ii = 1:size(scale_space,2)
    for jj=1:size(scale_space,1)
        mag_space{jj, ii} = zeros(size(scale_space{jj,ii},1),...
            size(scale_space{jj,ii},2));
        theta_space{jj, ii} = zeros(size(scale_space{jj,ii},1),...
            size(scale_space{jj,ii},2));
        I = scale_space{jj,ii};
        for xx=2:size(mag_space{jj, ii},1)-1
            for yy=2:size(mag_space{jj, ii},2)-1
                mag_space{jj, ii}(xx,yy) = ...
                    sqrt(((I(xx + 1, yy)-I(xx -1,yy))^2)...
                    + ((I(xx,yy+1)-I(xx,yy-1))^2));
                theta_space{jj, ii}(xx,yy) = ...
                    atan2d((I(xx,yy+1)-I(xx,yy-1))...
                    , (I(xx+1,yy)-I(xx-1,yy)));
                theta_space{jj, ii}(xx,yy) = ...
                    mod(theta_space{jj, ii}(xx,yy) + 360,360);
```

```
            end
        end
        mag_space{jj, ii} = padarray(mag_space{jj, ii},...
            [offset offset],'both');
        theta_space{jj, ii} = padarray(theta_space{jj, ii},...
            [offset offset],'both');
    end
end
%% Orientation Assignment
coord_final2 = [];
for oo=1:size(coord_final,1)
    hist_bin = zeros(1, 36);

    idx = coord_final(oo, 1);
    idy = coord_final(oo, 2);
    mag_I = mag_space{idx, idy};
    theta_I = theta_space{idx, idy};
    %Window size ~ 1.5*sigma
    winsize = 2*ceil(1.5*coord_final(oo,8));
    if mod(winsize,2) == 0
        winsize = winsize + 1;
    end
    gauss_filter = fspecial('gaussian',...
        [winsize winsize], 1.5*coord_final(oo,8));
    currx = coord_final(oo,3) ;%* (2^(coord_final(oo, 2) -idy));
    curry = coord_final(oo,4) ;%* (2^(coord_final(oo, 2) -idy));

    theta_win = theta_I(currx-(winsize-1)/2+offset:...
        currx+(winsize-1)/2+offset,...
        curry-(winsize-1)/2+offset:...
        curry+(winsize-1)/2+offset);
    mag_win = mag_I(currx-(winsize-1)/2+offset:...
        currx+(winsize-1)/2+offset,...
        curry-(winsize-1)/2+offset:...
        curry+(winsize-1)/2+offset);
```

```
    mag_win = mag_win .* gauss_filter;
    for ii = 1:size(theta_win,1)
        for jj = 1:size(theta_win,2)
            theta = mod(theta_win(ii,jj)+360,360);
            m = mag_win(ii,jj);
            hist_bin(1, floor(theta/10)+1) = ...
                hist_bin(1, floor(theta/10)+1) + m;
        end

    end
    maxEl = max(hist_bin);

    for aa = 1:length(hist_bin)
        if hist_bin(aa) >= 0.8*maxEl
            if aa-1 <= 0
                X = 0:2;
                Y = hist_bin([36,1,2]);
            elseif aa+1 > 36
                X = 35:37;
                Y = hist_bin([35,36,1]);
            else
                X = aa-1:aa+1;
                Y = hist_bin(aa-1:aa+1);
            end
            %Interpolation
            [p,S,mu]  = polyfit([Y(1) Y(2) Y(3)], [X(1) X(2) X(3)], 2);
            dir = polyval(p,Y(2),[],mu)*10;
            coord_final2 = ...
                [coord_final2; coord_final(oo,:) idx idy hist_bin(aa) dir];
        end
    end
end
disp(size(coord_final2,1));

%% Local Image Descriptors
```

```
w=4;% In David G. Lowe experiment,divide the area into 4*4.
feature_vec = zeros(size(coord_final2,1),w*w*8);
validpoints = zeros(size(coord_final2,1),2);
for oo = 1:size(coord_final2,1)
    theta_I = theta_space{coord_final2(oo, 10), coord_final2(oo, 11)};
    mag_I = mag_space{coord_final2(oo, 10), coord_final2(oo, 11)};

    subpixel_idx = floor(coord_final2(oo, 6));
    subpixel_idy = floor(coord_final2(oo, 7));
    G1 = fspecial('gaussian', [16 16], 8);
    theta = coord_final2(oo,13);

    % Lowe descriptor
    %New approach
%     coor = [subpixel_idx+offset; subpixel_idy+offset];
%     size_mag = [size(mag_I,1)/2; size(mag_I,2)/2];
%
%     theta = 360 - theta;
%     rotMag = imrotate(mag_I, theta);
%     rotTheta = imrotate(theta_I, theta);
%
%     rotMat = [cosd(theta) -sind(theta); sind(theta) cosd(theta)];
%     temp_coor = rotMat*(coor - size_mag);
%     rot_coor = temp_coor + size_mag;
%
%     difMat = [(size(rotMag,1) - size(mag_I,1))/2;...
%         (size(rotMag,2) - size(mag_I,2))/2];
%     rot_coor_final = rot_coor + difMat;
%     rot_row = round(rot_coor_final(1));
%     rot_col = round(rot_coor_final(2));
%
%     temp_vec = zeros(4,4,8);
%     %Create window
%     for i = 1:1:16
%         for j = 1:1:16
```

```
%         rowTrue = round(rot_row + j - 9);
%             colTrue = round(rot_col + i - 9);
%
%             %Check if points between boundaries
%             mag = rotMag(rowTrue, colTrue)* G1(j,i);
%             ori = mod(rotTheta(rowTrue, colTrue) + 360, 360);
%             temp_vec(ceil(i/4),ceil(j/4),floor(ori/45)+1) =...
%             temp_vec(ceil(i/4),ceil(j/4),floor(ori/45)+1) + mag;
%             end
%         end
%     end
%
%     count = 1;
%     for i = 1:1:4
%         for j = 1:1:4
%             feature_vec(oo,count:count+7) = temp_vec(i,j,:);
%             count = count + 8;
%         end
%     end

%AI Shack
    %Current approach
    theta_newwin = theta_I(subpixel_idx-7+offset:subpixel_idx+8+offset,...
        subpixel_idy-7+offset:subpixel_idy+8+offset);
    mag_newwin = mag_I(subpixel_idx-7+offset:subpixel_idx+8+offset,...
        subpixel_idy-7+offset:subpixel_idy+8+offset);
    mag_newwin = mag_newwin .* G1;

%      theta_newwin = imrotate(theta_newwin,360-theta,'crop');
%     mag_newwin = imrotate(mag_newwin, 360-theta ,'crop');
    count = 1;
    %For each window, jump 4 by 4
    for i = 1:w:16
        for j = 1:w:16
            theta_f = theta_newwin(i:(i+3), j:(j+3));
```

```
                mag_f = mag_newwin(i:(i+3), j:(j+3));

                %For each element in the window
                for k = 1:4
                    for l = 1:4
                        %Rotation dependence
                        theta_bin = theta_f(k,l) - theta + (180/8) + 360;
                        theta_bin = mod(theta_bin,360);

                        %Trilinear interpolation
                        feature_vec(oo, count+floor(theta_bin/45)) =...
                            feature_vec(oo, count+floor(theta_bin/45))...
                            + mag_f(k,l);

                    end
                end
                count = count + 8;
            end
        end


        norm = sqrt(sum(feature_vec(oo,:).^2));
        feature_vec(oo,:) = feature_vec(oo,:)./norm;

        %Threshold values above 0.2 -> Illumination Independence
        feature_vec(oo, feature_vec(oo,:)>0.2)=0.2;

        norm = sqrt(sum(feature_vec(oo,:).^2));
        feature_vec(oo,:) = feature_vec(oo,:)./norm;

        validpoints(oo,1) = (2^(coord_final2(oo,2)-2))*coord_final2(oo, 7);
        validpoints(oo,2) = (2^(coord_final2(oo,2)-2))*coord_final2(oo, 6);


    end
```

```
% save('feature2.mat','feature_vec');
% save('validpoints2.mat','validpoints');
end
```

.7 Appendix B

```
I = imread('dataset/landscape-a.jpg');
I_comp = imread('dataset/landscape-b.jpg');

% feature1 = load('feature1.mat');
% features1 = feature1.feature_vec;
% validpoints1 = load('validpoints1.mat');
% vpts1 = validpoints1.validpoints;

[features1, vpts1] = siftfeature(I);
[features2, vpts2] = siftfeature(I_comp);

% feature2 = load('feature2.mat');
% features2 = feature2.feature_vec;
% validpoints2 = load('validpoints2.mat');
% vpts2 = validpoints2.validpoints;

[indexPairs,matchmetric] = matchFeatures(features1,features2,'Unique',true,...
'MaxRatio',0.6);

matchedLoc1 = vpts1(indexPairs(:,1),:);
matchedLoc2 = vpts2(indexPairs(:,2),:);

figure;showMatchedFeatures(I,I_comp,matchedLoc1,matchedLoc2,'montage');
```

Bibliography

[1] U. Sinha, "Generating a feature."

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[3] "A few sketchy notes about sift."

[4] "Demo software: Sift keypoint detector."