1. **Mention the advantages and the disadvantages of the proposed method (selection of key points through the SIFT method)**

Advantages

- This method relies on the detection of unique features in an image and in a sense creates a digital signature for an object. Not only are the features robust (reproducible) in spite of rotation or scale but they also resist minor changes to illumination, noise, minor occlusions and changes in the viewing angle.
- One of the strongest advantages is that the algorithm allows us to store the descriptors in a file and create a database of potentially recognizable objects which allows reusability and helps save time.
- The paper also says that as few as 3 SIFT features from an object are enough to compute its location and pose i.e. for reliable object recognition. This would allow an object that is well occluded to also be detected.
- Detection of key features across slight changes in viewing angles allows many useful practical applications like creation of 3D maps, motion tracking and image panorama assembly.

Disadvantages

- It is possible that 2 different objects may have the same set of features. Although the probability is almost negligible, it is technically possible for objects that are similar.
- The paper says that this approach of recognition offers near real-time performance. The paper does not justify this claim and based on the complexity of the algorithm it seems unlikely for most realistic scenarios.
- This technique is fairly complex to understand and implement and may require a lot of time and effort in tweaking given that it has a number of parameters. It really isn't a quick and simple approach.
- A disadvantage of this algorithm is that it does not restrict the feature selection to a particular object in the image but utilizes the entire image itself. This means that in a

cluttered image, feature from the background may interfere and may result in false results.

2. How will you use this method for vision-based robot navigation? Please explain.

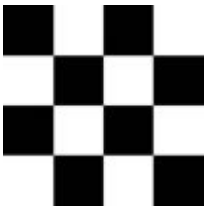
- When a robot is placed in an alien environment one of the biggest challenges is to localize its position and estimate a map of its surroundings. SIFT can be used towards Simultaneous Localization and Mapping (SLAM) by using the features from successive images to stitch a map of its surroundings. By estimating the depth at periodic angles, the robot would be able to generate a 3D map of its surroundings and localize itself.
- A robot can also know for example the exact way to go through a door by gathering the SIFT features which would indicate the extreme corners. In other words, SIFT could be used by a robot to align itself to move in the right direction.
- Using SIFT features, a robot can continuously track a slow moving object as long as the conditions related to luminosity, noise, occlusion, etc. are within limits. Given the computational requirements, this may be difficult for an object that is moving relatively fast.

3. Explain the computational implications of the method

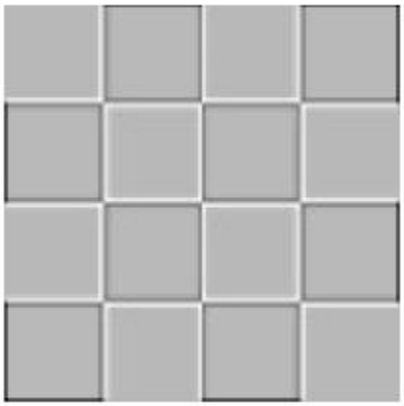
- Although the paper states that the algorithm can achieve close to real time performance with modern computers and with a small database, it seems unlikely that this claim would be useful for practical applications.
- Another aspect is that given the complexity of the implementation it seems difficult to reach a level of optimization required to reach close to real time performance both in terms of memory and time.
- By using the cascade filtering approach, the cost of feature extraction is minimized significantly as some of the more expensive tests are applied to probable key points that clear a certain stage. Thus, by design the algorithm is fairly optimized.

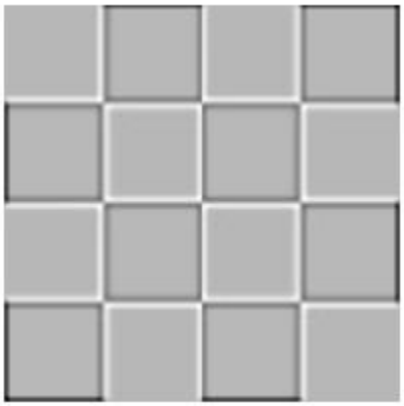
4. Write some code that selects some features through the SIFT method

Original Image (100 x 100)

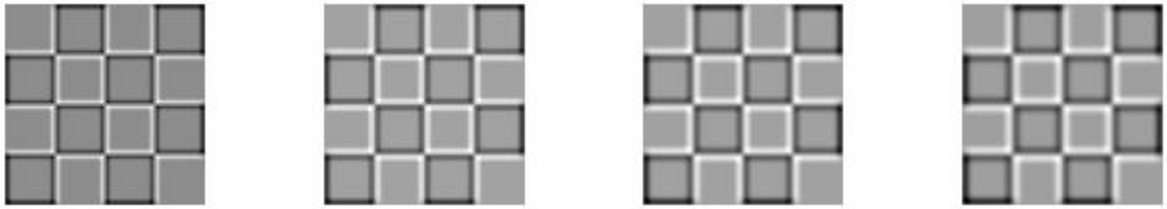


Difference of Gaussians – Octave 1 (200 x 200)





Octave 2 (100 x 100px)

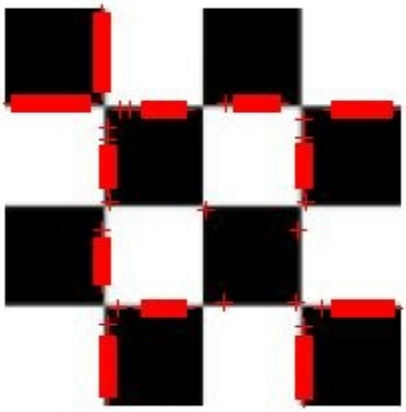


Octave 3 (50 x 50px)

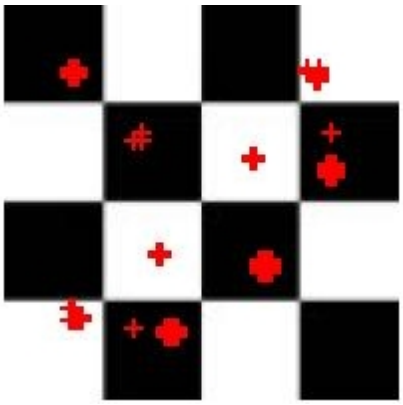
Octave 4 (25 x 25px)

Improper scaling

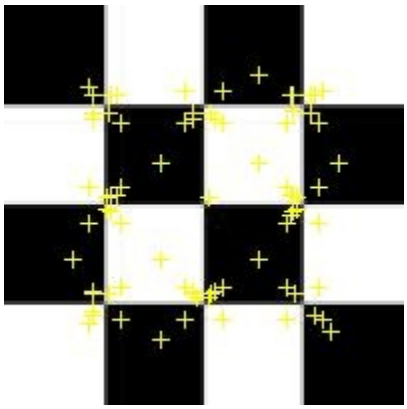
Initially I wasn't scaling my results correctly which led to a clustering of points near borders



Final result



Expected result - run on sample code



Although the final result is not quite what is expected it is because of the way I have implemented the merging of keypoints across scales. My approach is to expand the matrix of keypoints by padding each cell by itself by an 'x' number of times where 'x' is the scaling factor. This leads to clustering of points around a region which I should be able to remove quite easily.

Although the pattern in the two images is similar, I believe the missing / additional keypoints in my implementation are primarily due to the lack of proper scaling.

Code

```
%-------------------------------------------------------------------------
% Pushkar Modi
% Computer Vision - Assignment 3
% Detecting objects in an image using SIFT
%
% Reference:
% David G. Lowe, "Distinctive Image Features from Sacle-Invariant Keypoints",
% accepted for publication in the International Journal of Computer
% Vision, 2004.
%-------------------------------------------------------------------------

function SIFT()

close all;
clear all;
clc;

noOfOctaves = 4;
s = 2;

I = imread('grid.jpg');
I = im2double(I);
I = rgb2gray(I);
I = scale(I);


% STEP 1 - DIFFERENCE OF GAUSSIANS
% This step requires that we take an image, blurred at different gaussian
% levels and find the difference between adjacent blurred Gaussian images.
% A set of  blurs on the same scale form an octave.
% We repeat this step for octaves of the same image at different scales.

% Pre-processing Step
gaussianFilter = fspecial('gaussian', [3, 3], 0.5);
blurredI = imfilter(I, gaussianFilter, 'replicate', 'conv');
I = doubleImage(I);

keypointsFromOctaves = buildOctaves(I, s, noOfOctaves);
keypoints = keypointsFromOctaves(1).keypoints;
keypoints =  padarray(keypoints, [1 1], 'post');

for i = 2 : noOfOctaves
    scaledPoints = mapKeypoints(keypointsFromOctaves(i).keypoints, i-1);
    keypoints = keypoints + scaledPoints;
end

figure;
imshow(I);
plotKeypoints(keypoints);
```

```
function octave = buildOctaves(I, s, noOfOctaves)

k = 2^(1 / s);
sigma = 1.6;
scalesInOctave = s + 3;

for oct = 1 : noOfOctaves

    octave(oct).image = I;
    for i = 1 : scalesInOctave

        sigma_g = (k^i) * sigma;
        gaussianFilter = fspecial('gaussian', [3, 3], sigma_g);

        if (i==1)
            octave(oct).slice(:, :, i) = scale(imfilter(I, gaussianFilter));
        else
            octave(oct).slice(:, :, i) = scale(imfilter(octave(oct).slice(:,
:, i-1), gaussianFilter));
            octave(oct).dog(:, :, i-1) = scale(octave(oct).slice(:, :, i) -
octave(oct).slice(:, :, i-1));

%             figure;
%             imshow(octave(oct).dog(:, :, i-1));
        end
    end

    [m, n] = size(I);
    scaleKeyPoints = zeros(m, n);

    for i = 2 : scalesInOctave - 2  %i is the ith DOG image
        for row = 2 : m-1
            for col = 2 : n-1
                M = [];
                a = i - 1;
                b = i + 1;

                M = (octave(oct).dog(row-1:row+1, col-1:col+1, a:b));
                MaxM = max(max(max(M)));
                MinM = min(min(min(M)));

                if (( MaxM == M(2, 2, 2) || MinM == M(2, 2, 2) ) && MaxM ~=
MinM)
                    scaleKeyPoints(row, col) = 1;
                end
            end
        end
    end

    octave(oct).keypoints = filterKeypoints(scaleKeyPoints, scalesInOctave-1,
octave(oct).dog, octave(oct).image);
    figure;
    imshow(octave(oct).image);
    plotKeypoints(octave(oct).keypoints);
    size(find(octave(oct).keypoints))
```

```
    if(oct < 4)
        I = reduceImage(octave(oct).slice(:, :, 1));
    end
end
```

```
function out = filterKeypoints(scaleKeyPoints, noOfDOGs, dog, I)

[m,n] = size(I);
out = zeros(size(scaleKeyPoints));

for i = 2 : noOfDOGs - 1  %i is the ith DOG image
    for row = 2 : m-1
        for col = 2 : n-1

            if (scaleKeyPoints(row, col) ~= 1)
                continue;
            end

            r=row; c=col; d=i;

            for iter=1:5
                if((r<=1 || c<=1) || d<=1)
                    break;
                end
                if((r>=m-1 || c>=n-1) || d>=noOfDOGs)
                    break;
                end

                sp = dog(r-1:r+1, c-1:c+1, d-1:d+1);
                [H] = scale(hessian(sp));

                % If the matrix is not invertible delx set to zero
                if( rcond(H) < 1e-10 )
                    delx = 0;
                else
                    delx = -inv(H)*derivative(sp);
                end

                if(round(norm(delx)) == 0)
                    Dmag = dog(r,c,d) + 0.5 * delx' * derivative(sp);
                    curvature_ratio = getEdgeResponse(sp);

                    if((Dmag > 0.3) & (curvature_ratio < 10))
                        out(r, c) = 1;
                        break;
                    end
                else
                    r = r + round(delx(1));
                    c = c + round(delx(2));
                    d = d + round(delx(3));
                end
```

```
            end
        end
    end
end


% This funcion plots keypoints on the original image
function plotKeypoints(scaleKeyPoints)

hold on;
[xind,yind] = find(scaleKeyPoints >= 2);
plot(xind, yind, 'r+');



% This function doubles the size of the given image by linear interpolation
function out =  doubleImage(I)

[m, n] = size(I);
[X, Y] = meshgrid( 1: 0.5 : n, 1: 0.5: m );
out = interp2(I, X, Y, '*linear');



% Take the 2nd row and column for the next scale
function out = reduceImage(I)

[m, n] = size(I);
out = I(1:2:m, 1:2:n);



% Here SP is 3x3x3 matrix of candidate keyPoints
function out = hessian(sp)

Dxx = dd1(sp(2,:,2));
Dxy = dd2(sp(:,:,2));
Dyx = Dxy;
Dxd = dd2([sp(2,:,1); sp(2,:,2); sp(2,:,3)]);
Ddx = Dxd;
Dyy = dd1(sp(:,2,2));
Dyd = dd2([sp(:,2,1)  sp(:,2,2)  sp(:,2,3)]);
Ddy = Dyd;
Ddd = dd1([sp(2,2,1) sp(2,2,2) sp(2,2,3)]);
out = [Dxx Dxy Dxd; Dyx Dyy Dyd; Ddx Ddy Ddd];



function out = getEdgeResponse(sp)

Dxx = dd1(sp(2,:,2));
Dyy = dd1(sp(:,2,2));
Dxy = dd2(sp(:,:,2));
traceH = Dxx + Dyy;
detH = (Dxx * Dyy) - (Dxy * Dxy);
```

```
if (detH ~= 0)
    ratio = traceH^2 / detH;
else
    ratio = 0;
end
out = ratio;
```

```
function out = derivative(sp)

Dy = (sp(3,2,2) - sp(1,2,2)) / 2;
Dx = (sp(2,3,2) - sp(2,1,2)) / 2;
Dd = (sp(2,2,3) - sp(2,2,1)) / 2;
out = [Dx; Dy; Dd];
```

```
% Function for single derivative
function out = dd1(f)

i = 2;
out = ( f(i+1) - f(i-1) ) / 2;
```

```
% Function for double derivative
function out = dd2(f)

i = 2;
j = 2;
out = ( f(i+1,j-1) + f(i-1,j+1) - f(i-1,j-1) - f(i+1,j+1) ) / 4;
```

```
function out = mapKeypoints(D, factor)

for f = 1 : factor
    [m,n] = size(D);
    out = [];

    for i=1:m
        A = [];
        for j=1:n
            cells = padarray(D(i,j), [1,1], 'replicate', 'post');
            A = horzcat(A, cells);
        end
        out = vertcat(out, A);
    end

    D = out;
End
```

```
% This function scales the values of an image to between 0 & 1

function output = scale(inputImage)

minValue = min(inputImage);             % returns row as a vector
minValue = min(minValue);               % returns max value from the row
inputImage = inputImage - minValue;     % This will scale the lowest value to
0

maxValue = max(inputImage);             % returns row as a vector
maxValue = max(maxValue);               % returns max value from the row
inputImage = inputImage / maxValue;     % This will scale the highest value
to 1

output = inputImage;
```

5. Reference

- David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", accepted for publication in the International Journal of Computer Vision, 2004.