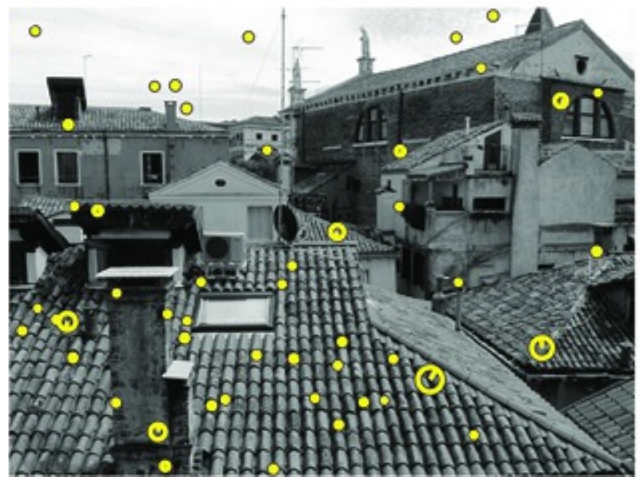
Laboratory of Image Processing

SIFT & MatLab



Pier Luigi Mazzeo

pierluigi.mazzeo@cnr.it

Institute of Applied Sciences and Intelligent Systems

ScienceApp

National Research Council of Italy

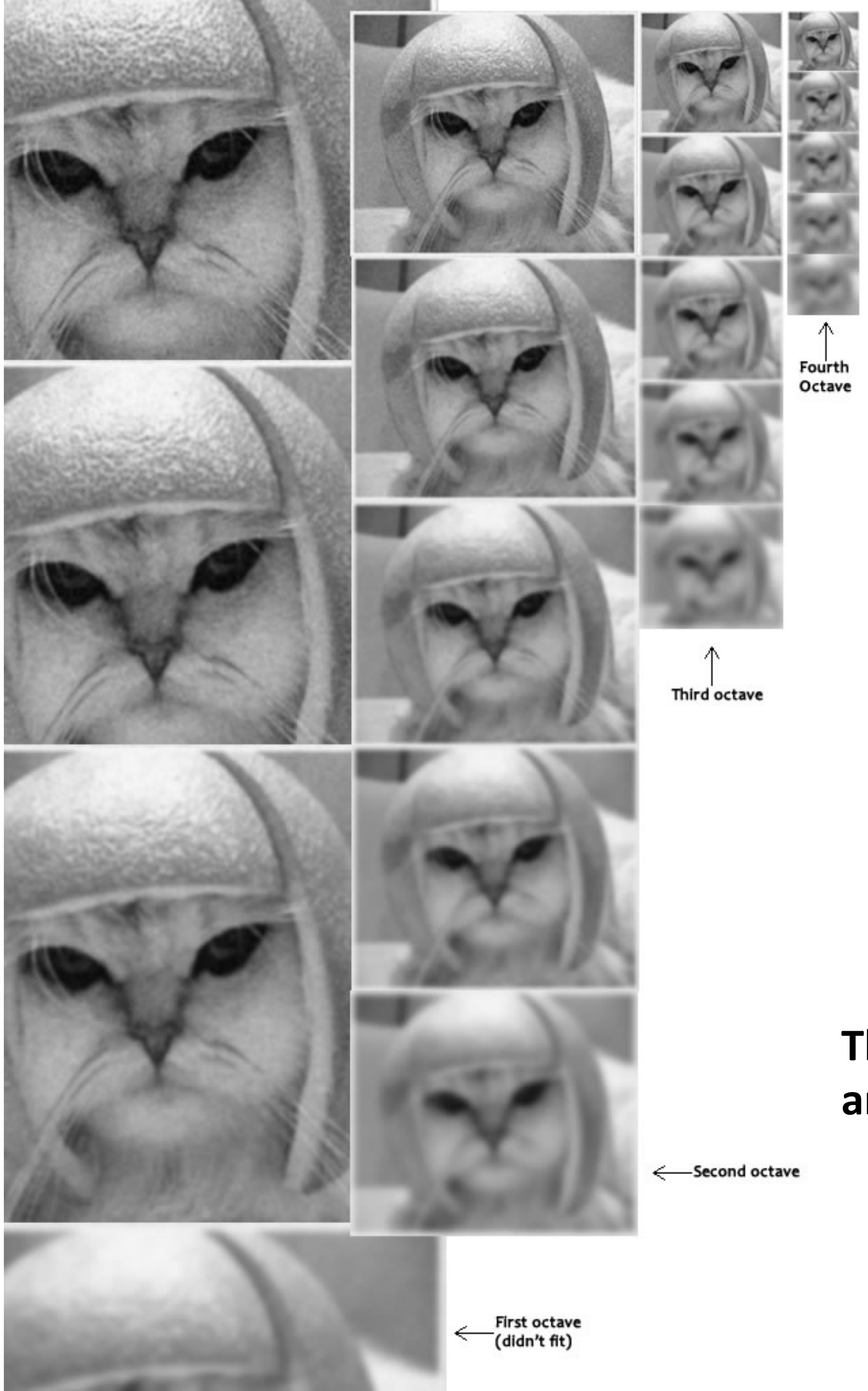
Sift purpose

- Find and describe interest points invariants to:
  - Scale
  - Rotation
  - Illumination
  - Viewpoint

Do it Yourself

- Constructing a scale space
- LoG Approximation
- Finding keypoints
- Get rid of bad key points (A technique similar to the Harris Corner Detector)
- Assigning an orientation to the keypoints
- Generate SIFT features

Construction of a scale space



SIFT takes scale spaces to the next level. You take the original image, and generate progressively blurred out images. Then, you resize the original image to half size. And you generate blurred out images again. And you keep repeating.

The creator of SIFT suggests that 4 octaves and 5 blur levels are ideal for the algorithm

Construction of a scale space (details)

- The first octave
- If the original image is doubled in size and antialiased a bit (by blurring it) then the algorithm produces more four times more keypoints. The more the keypoints, the better!
- Blurring

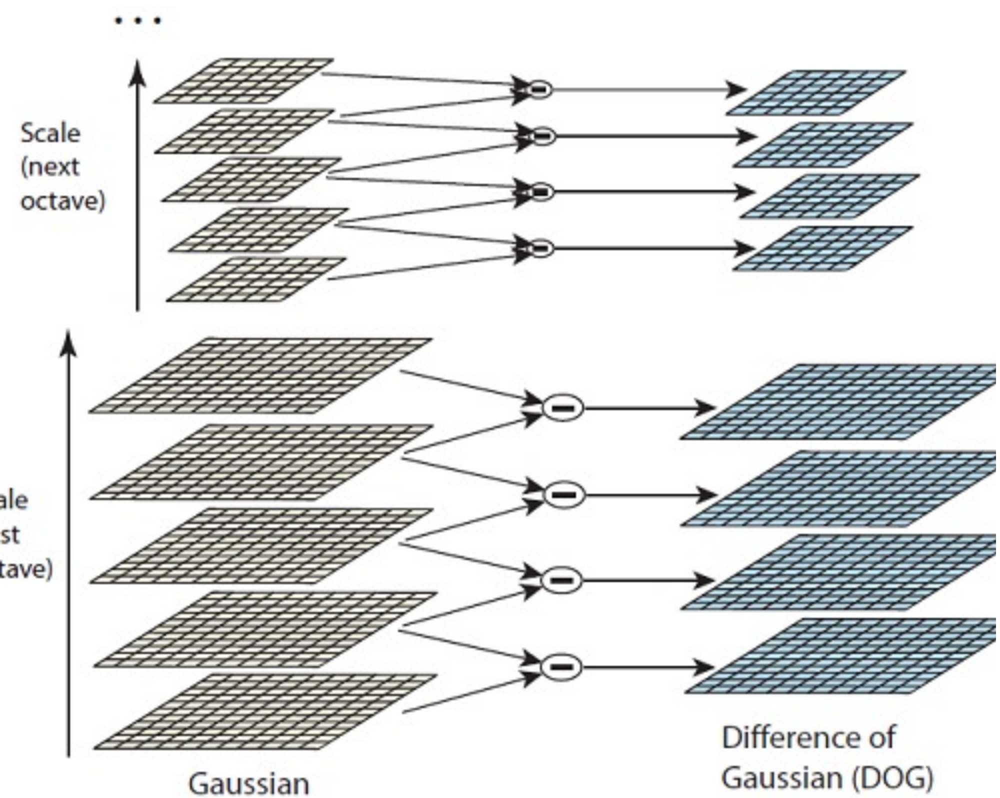
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

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

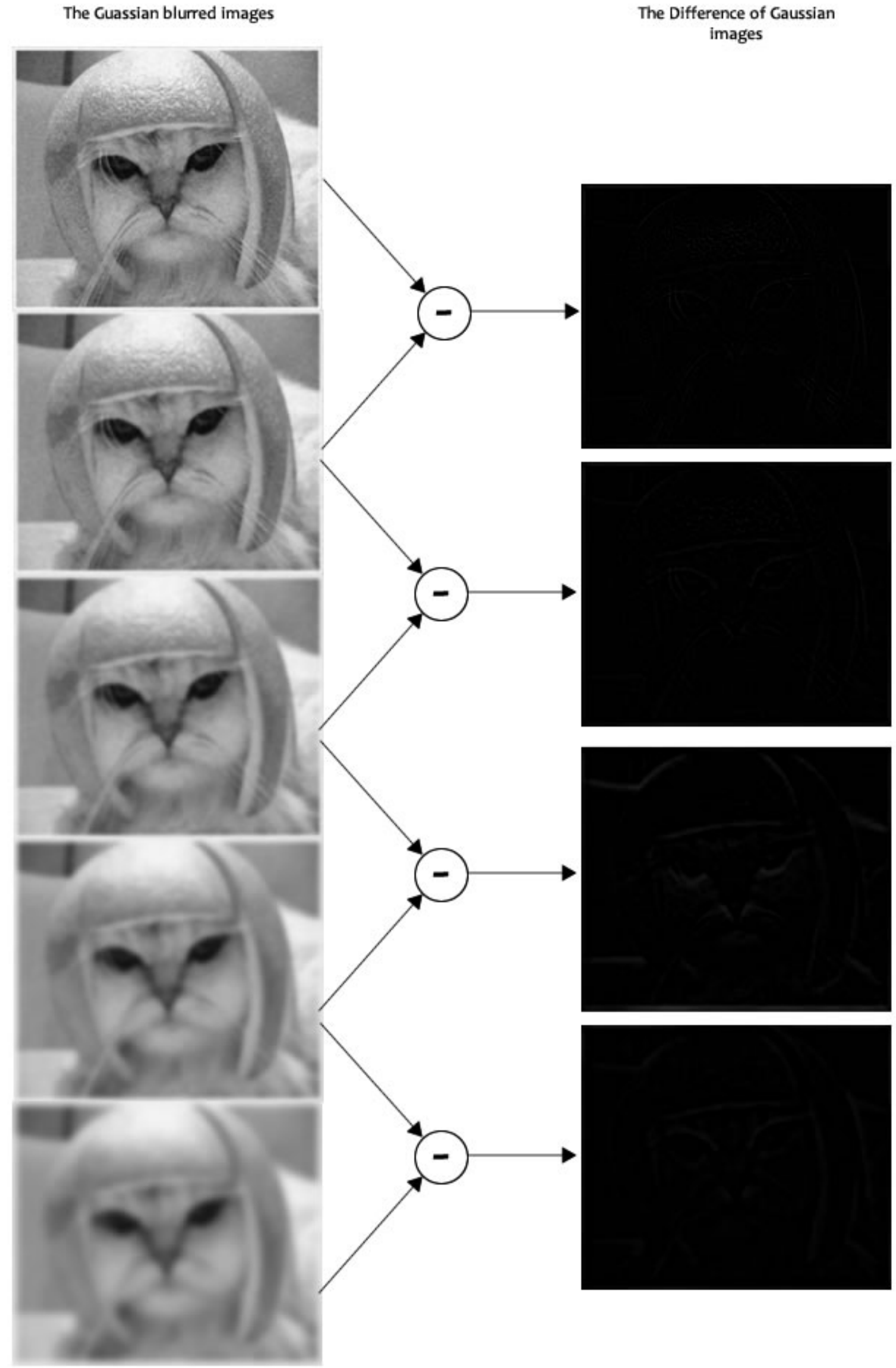
- Amount of Blurring

octave \ scale →				
0.707107	1.000000	1.414214	2.000000	2.828427
1.414214	2.000000	2.828427	4.000000	5.656854
2.828427	4.000000	5.656854	8.000000	11.313708
5.656854	8.000000	11.313708	16.000000	22.627417

LoG approximation



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



Design Gaussian filter

```
% gauss_filter: Obtains a smoothed gaussian filter image
%    - Output:
%        smooth: image filter by a gaussian filter
%    - Input:
%        image:  Matrix containing single band image.
%        sigma:  Value of the sigma for the Gaussian filter
%        kernel_size: Size of the gaussian kernel (default: 6*sigma)
function [smooth]= gauss_filter(image,sigma,kernel_size)
if nargin < 2
    sigma=1;
end
if nargin < 3
    kernel_size=6*sigma;
end
gaussian_radio=floor(kernel_size/2);  %radio of the gaussian
x=[-gaussian_radio : gaussian_radio];   % x values (gaussian kernel size)
gaussiano= exp(-x.^2/(2*sigma^2))/(sigma*sqrt(2*pi) );   %calculate the
unidimensional gaussian
%gaussiano=round(gaussian/min(abs(nonzeros(gaussian))));  %aproximate the
unidimensional kernel to integers
temp=conv2(double(image),double(gaussiano),'same');
smooth=conv2(double(temp),double(gaussiano'),'same');
end
```

SIFT Matlab Implementation

```
% Obtain the SIFT interest points, and calculate the SIFT descriptor.
% INPUTS
%   image: Image to look for interest points. The image is in gray scale.
%   num_octaves: Number of octaves
%   num_intervals: Number of intervals s in to achieve the double of the sigma
%   value. In the original paper s=2
%   sigmavalue: Value of the initial sigma
% OUTPUTS
%   sift_desc: matrix containing in each row an intersting point
%   [x1 y1  octave  num_imagen_in_the_scale  sigma_of_the_scale  orientation(bins) ]
function [sift_desc]=sift(image,num_octaves,num_intervals,sigmavalue)

[dimx,dimy]=size(image);
%define constants of the application
antialiassigma=0.5;
k1=2^(1/num_intervals);
r_curvature=5;
contrast_threshold=0.03; %Threshold determining a low contrast keypoint to be eliminated

if sigmavalue>1.1
 % contrast_threshold=0.07;
end
if sigmavalue==1.4
    contrast_threshold=0.06;
End
updown=1;
```

SIFT Matlab Implementation: impyramid

`Impyramid`: implement Image pyramid reduction and expansion. If A is M-by-N and DIRECTION is 'reduce', then the size of B is ceil(M/2)-by-ceil(N/2). If DIRECTION is 'expand', then the size of B is (2*M-1)-by-(2*N-1).

```
I0 = imread('cameraman.tif');
I1 = impyramid(I0, 'expand');
I2 = impyramid(I1, 'reduce');
I3 = impyramid(I2, 'reduce');
imshow(I0)
figure, imshow(I1)
figure, imshow(I2)
figure, imshow(I3)
```

DOG: Example

```
%Allocate memory
gaussians = cell(4); dogs=cell(4);
%Expansions
init_image=impyramid(gauss_filter(I0,0.5,42),'expand');
%Gaussians filtering first interval
gaussians(1)={gauss_filter(init_image,1,4)};
previmage=cell2mat(gaussians(1)) %Obtain the previous image
imagesc(previmage);
%Gaussians filtering second interval
newimage=gauss_filter(previmage,2,8); %apply a new smoothing
Dog=newimage-previmage; %calculate the difference of gaussians
Gaussians(2)={newimage}; %Store the results
dogs(1)={Dog};
imagesc(Dog);
```

SIFT Matlab Implementation: allocate memory

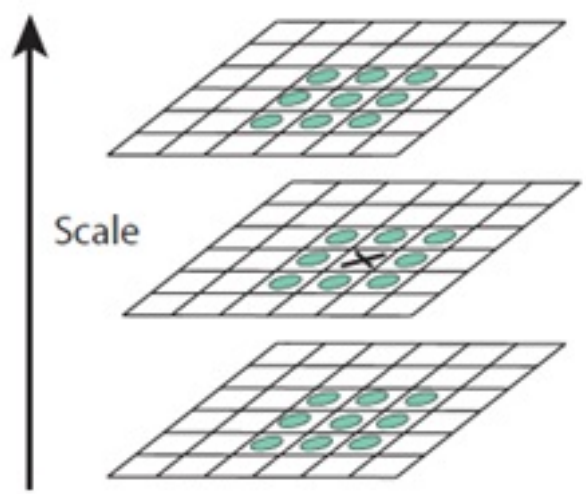
```
%normalize image input
% Assuming 8 bits image
image1=double(image)/255;


% %%%%%%% Allocate memory %%%%%%%%%
gaussians = cell(num_intervals+3,num_octaves);
dogs=cell(num_intervals+2,num_octaves);
magnitude=cell(num_intervals,num_octaves);
orientation=cell(num_intervals,num_octaves);
for i=1:num_octaves
   for j=1:num_intervals+3
      gaussians(j,i)= {zeros(dimx/(2^(i-2)),dimy/(2^(i-2)))};
   end
   for j=1:num_intervals+2
      dogs(j,i)= {zeros(dimx/(2^(i-2)),dimy/(2^(i-2)))};
   end
   for j=1:num_intervals
       magnitude(j,i)=  {zeros((dimx/(2^(i-2)))-2,(dimy/(2^(i-2)))-2)};
       orientation(j,i)={zeros((dimx/(2^(i-2)))-2,(dimy/(2^(i-2)))-2)};
   end
end
```

SIFT Matlab Implementation: DOG

```
% %%% Create first interval of the first octave %%%%%
init_image=impyramid(gauss_filter(image1,antialiassigma,4*antialiassigma),'expand');
gaussians(1)={gauss_filter(init_image,sigmavalue,4*sigmavalue)};

 % %%% Generates all the blurred out images for each octave %%%%
% %%%               and the DoG images                     %%%%
for i=1:num_octaves
   sigma=sigmavalue;    %reset the sigma value
   for j=1:(num_intervals+2)
       sigma=sigma*2^((j-1)/2);     %Assign a sigma value acording to the scale
       previmage=cell2mat(gaussians(j,i));  %Obtain the previous image
       newimage=gauss_filter(previmage,sigma,4*sigma);  %apply a new smoothing
       dog=previmage-newimage;  %calculate the difference of gaussians

       %save the results
       gaussians(j+1,i)={newimage};
       dogs(j,i)={dog};
   end

   %Build the  init image in the next level
   if(i<num_octaves)
       lowscale=cell2mat(gaussians(num_intervals+1,i));
       upscale=impyramid(lowscale,'reduce');
       gaussians(1,i+1)={upscale};
   end
end
```

Finding keypoints

- a) Locate maxima/minima in DoG images



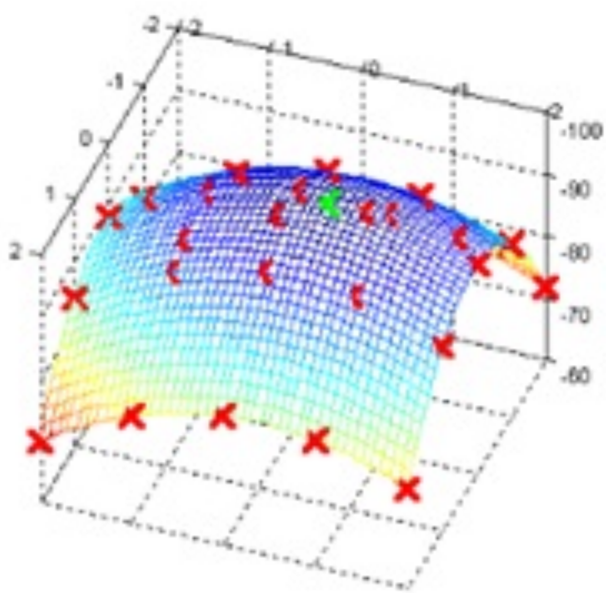
Local extrema detection, the pixel marked × is compared against its 26 neighbors in a 3 × 3 × 3 neighborhood that spans adjacent DoG images.

Matlab Implementation (find local extrema)

```
list_points=[];
for i=1:num_octaves
    for j=2:(num_intervals+1)
        % Obtain the matrices where to look for the extrema
        level=cell2mat(dogs(j,i));
        up=cell2mat(dogs(j+1,i));
        down=cell2mat(dogs(j-1,i));

        [sx,sy]=size(level);

        %look for a local maxima
        local_maxima=(level(2:sx-1,2:sy-1)>level(1:sx-2,1:sy-2)) & ( level(2:sx-1,2:sy-1) > level(1:sx-2,2:sy-1) ) &
      (level(2:sx-1,2:sy-1)>level(1:sx-2,3:sy)) & (level(2:sx-1,2:sy-1)>level(2:sx-1,1:sy-2)) &
      (level(2:sx-1,2:sy-1)>level(2:sx-1,3:sy)) & (level(2:sx-1,2:sy-1)>level(3:sx,1:sy-2)) & (level(2:sx-1,2:sy-1)>level(3:sx,2:sy-1))
      & (level(2:sx-1,2:sy-1)>level(3:sx,3:sy)) ;
        local_maxima=local_maxima & (level(2:sx-1,2:sy-1)>up(1:sx-2,1:sy-2)) & ( level(2:sx-1,2:sy-1) > up(1:sx-2,2:sy-1) ) &
      (level(2:sx-1,2:sy-1)>up(1:sx-2,3:sy)) & (level(2:sx-1,2:sy-1)>up(2:sx-1,1:sy-2)) & (level(2:sx-1,2:sy-1)>up(2:sx-1,2:sy-1)) &
      (level(2:sx-1,2:sy-1)>up(2:sx-1,3:sy)) & (level(2:sx-1,2:sy-1)>up(3:sx,1:sy-2)) & (level(2:sx-1,2:sy-1)>up(3:sx,2:sy-1)) &
      (level(2:sx-1,2:sy-1)>up(3:sx,3:sy)) ;
        local_maxima=local_maxima & (level(2:sx-1,2:sy-1)>down(1:sx-2,1:sy-2)) & ( level(2:sx-1,2:sy-1) > down(1:sx-2,2:sy-1) ) &
      (level(2:sx-1,2:sy-1)>down(1:sx-2,3:sy)) & (level(2:sx-1,2:sy-1)>down(2:sx-1,1:sy-2)) & (level(2:sx-1,2:sy-1)>down(2:sx-1,2:sy-1))
      & (level(2:sx-1,2:sy-1)>down(2:sx-1,3:sy)) & (level(2:sx-1,2:sy-1)>down(3:sx,1:sy-2)) & (level(2:sx-1,2:sy-1)>down(3:sx,2:sy-1)) &
      (level(2:sx-1,2:sy-1)>down(3:sx,3:sy)) ;

        %look for a local minima
        local_minima=(level(2:sx-1,2:sy-1)<level(1:sx-2,1:sy-2)) & ( level(2:sx-1,2:sy-1) < level(1:sx-2,2:sy-1) ) &
      (level(2:sx-1,2:sy-1)<level(1:sx-2,3:sy)) & (level(2:sx-1,2:sy-1)<level(2:sx-1,1:sy-2)) &
      (level(2:sx-1,2:sy-1)<level(2:sx-1,3:sy)) & (level(2:sx-1,2:sy-1)<level(3:sx,1:sy-2)) & (level(2:sx-1,2:sy-1)<level(3:sx,2:sy-1))
      & (level(2:sx-1,2:sy-1)>level(3:sx,3:sy)) ;
        local_minima=local_minima & (level(2:sx-1,2:sy-1)<up(1:sx-2,1:sy-2)) & ( level(2:sx-1,2:sy-1) < up(1:sx-2,2:sy-1) ) &
      (level(2:sx-1,2:sy-1)<up(1:sx-2,3:sy)) & (level(2:sx-1,2:sy-1)<up(2:sx-1,1:sy-2)) & (level(2:sx-1,2:sy-1)<up(2:sx-1,2:sy-1)) &
      (level(2:sx-1,2:sy-1)<up(2:sx-1,3:sy)) & (level(2:sx-1,2:sy-1)<up(3:sx,1:sy-2)) & (level(2:sx-1,2:sy-1)<up(3:sx,2:sy-1)) &
      (level(2:sx-1,2:sy-1)<up(3:sx,3:sy)) ;
        local_minima=local_minima & (level(2:sx-1,2:sy-1)<down(1:sx-2,1:sy-2)) & ( level(2:sx-1,2:sy-1) < down(1:sx-2,2:sy-1) ) &
      (level(2:sx-1,2:sy-1)<down(1:sx-2,3:sy)) & (level(2:sx-1,2:sy-1)<down(2:sx-1,1:sy-2)) & (level(2:sx-1,2:sy-1)<down(2:sx-1,2:sy-1))
      & (level(2:sx-1,2:sy-1)<down(2:sx-1,3:sy)) & (level(2:sx-1,2:sy-1)<down(3:sx,1:sy-2)) & (level(2:sx-1,2:sy-1)<down(3:sx,2:sy-1)) &
      (level(2:sx-1,2:sy-1)<down(3:sx,3:sy)) ;

        extrema=local_maxima | local_minima;
  %INSERT THE PART OF RID BAD KEY POINT
  end
```

Finding keypoints

- b) Find subpixel maxima/minima



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

Get rid of bad key points

- Removing low contrast features

If the magnitude of the intensity (i.e., without sign) at the current pixel in the DoG image (that is being checked for minima/maxima) is less than a certain value, it is rejected

- Removing edges

Tr (H) = Dxx + Dyy

Det(H) = DxxDyy - (Dxy)2

$$R = \mathrm{Tr}(H)^2 / \mathrm{Det}(H)$$

- If the value of R is greater for a candidate keypoint, then that keypoint is poorly localized and hence rejected.

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

$$\frac{D_{xx} + D_{yy}}{D_{xx} D_{yy} - (D_{xy})^2} < \frac{(r+1)^2}{r}$$

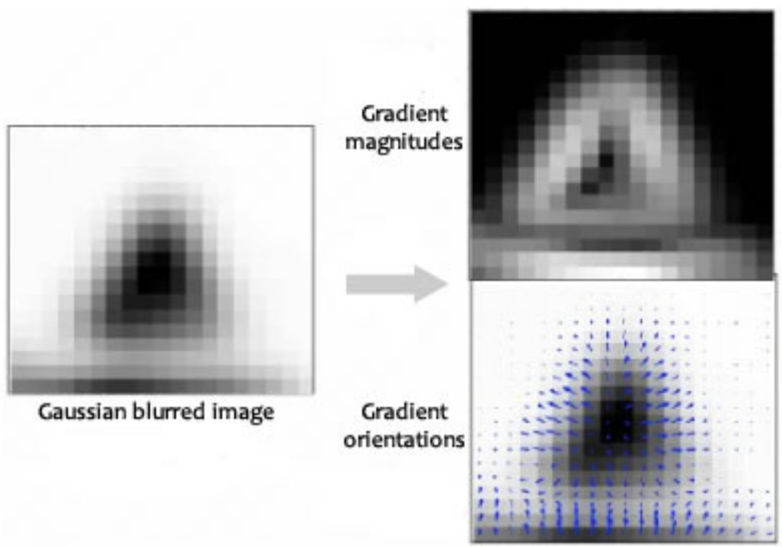
Matlab Implementation (remove poorly contrasted keypoints)

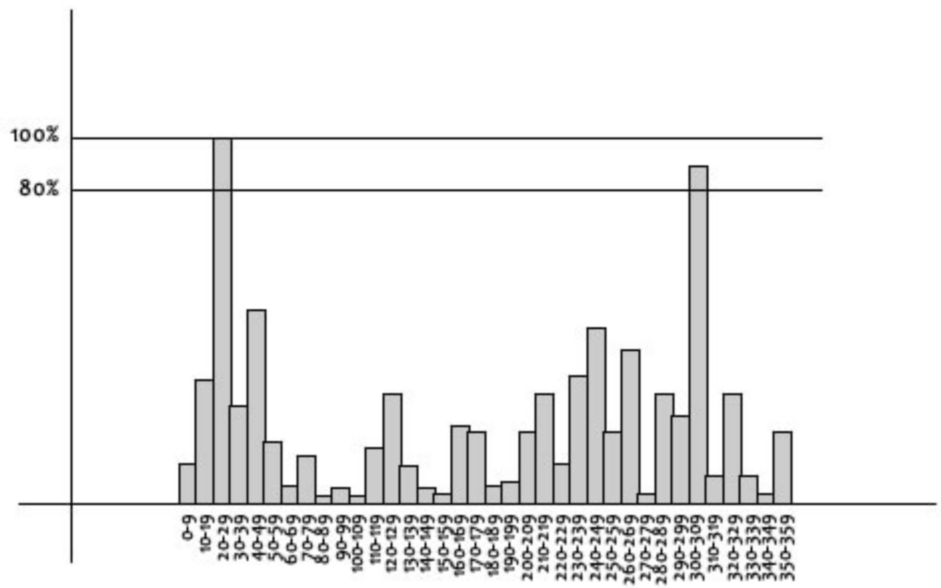
```
%indices of the extrema points
        [x,y]=find(extrema);
        numtimes=size(find(extrema));

        for k=1:numtimes
            x1=x(k);
            y1=y(k);
            if(abs(level(x1+1,y1+1))<contrast_threshold)    %low contrast point are discarded
               extrema(x1,y1)=0;
            else   %keep being extrema, check for edge
               rx=x1+1;
               ry=y1+1;
               fxx= level(rx-1,ry)+level(rx+1,ry)-2*level(rx,ry);   % double derivate in x direction
               fyy= level(rx,ry-1)+level(rx,ry+1)-2*level(rx,ry);   % double derivate in y direction
               fxy= level(rx-1,ry-1)+level(rx+1,ry+1)-level(rx-1,ry+1)-level(rx+1,ry-1); %derivate inx
    and y direction
               trace=fxx+fyy;
               deter=fxx*fyy-fxy*fxy;
               curvature=trace*trace/deter;
               curv_threshold= ((r_curvature+1)^2)/r_curvature;
               if(deter<0 || curvature>curv_threshold)   %Reject edge points
                   extrema(x1,y1)=0;
               end
            end
        end
%check
list_points=[list_points; [x y repmat(i,numtimes,1) repmat(j-1,numtimes,1) zeros(numtimes,1)
zeros(numtimes,1)] ];
```

Plot KP-Position

```
 list_points= sift_desc;
  figure; imshow(image,[]);hold on;
  x3=[];
  y3=[];
      for j=1:size(list_points,1),
          x4=list_points(j,2)*2^(list_points(j,3)-2);
          y4=list_points(j,1)*2^(list_points(j,3)-2);
          plot(x4 ,y4 , 'ro', 'color', [1 0 0], 'markerfacecolor', [1
0 0], 'markersize', 2.5 );
          x3=[x3;x4];
          y3=[y3;y4];
      end
      hold off
```

Assigning an orientation to the keypoints

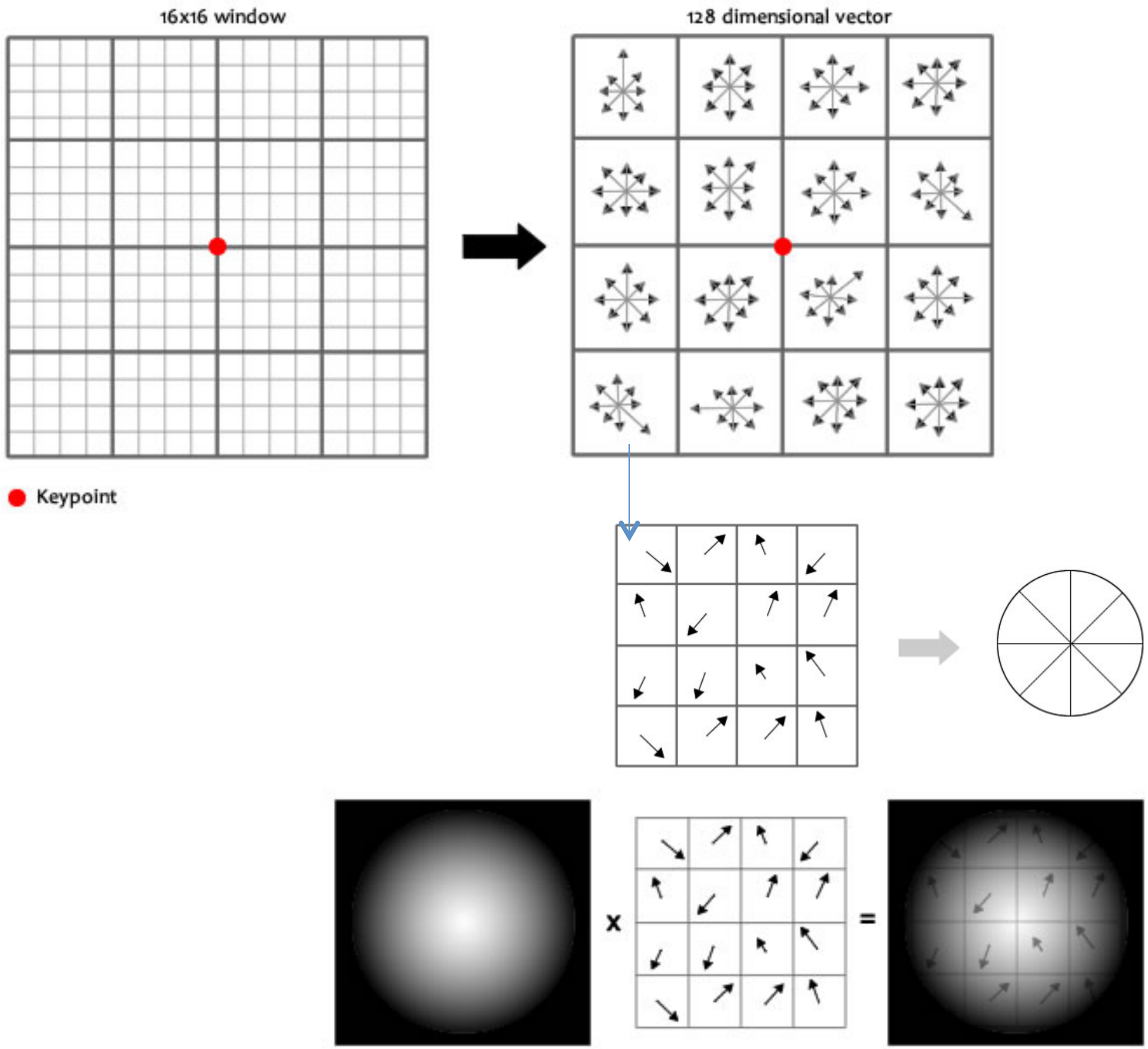




$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

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

Generate SIFT features



Generate SIFT features

- You take a 16×16 window of “in-between” pixels around the keypoint. You split that window into sixteen 4×4 windows. From each 4×4 window you generate a histogram of 8 bins. Each bin corresponding to 0-44 degrees, 45-89 degrees, etc. Gradient orientations from the 4×4 are put into these bins. This is done for all 4×4 blocks. Finally, you normalize the 128 values you get.
- To solve a few problems, you subtract the keypoint’s orientation and also threshold the value of each element of the feature vector to 0.2 (and normalize again).

Matlab Implementation (KP-calculate magnitude and orientation)

```
% %%%%  generate an orientation for all the key points  %%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculate magnitude,orientation
for i=1:num_octaves
    for j=1:num_intervals
        previmage=cell2mat(gaussians(j+updown,i));   %add 2 to
associate to the sigma of the upper image, or add 1 to associate to
the sigma of the lower one.
        [rx,ry]=size(previmage);
        dx=previmage(3:rx,2:ry-1)-previmage(1:rx-2,2:ry-1);
        dy=previmage(2:rx-1,1:ry-2)-previmage(2:rx-1,3:ry);
        mag=sqrt(dx.*dx + dy.*dy);
        ori=atan2(dy,dx);
        magnitude(j,i)={mag};
        orientation(j,i)={ori};
    end
end
```

GaussWindow Function

```
function [f,n] = gausswindow( s,n )
%gausswindow: Create a gaussian window
with kernel size given by n, and
%sigma given by s
%   Detailed explanation goes here
if nargin<2
n=ceil(2*s); % force creation of options
end
x=-n:n;
[Y,X] = meshgrid(x,x);
f = exp( -(X.^2+Y.^2)/(2*s^2) );
end
```

Matlab Implementation (Calculate the KP-dominant orientation)

```
new_ones=[];

% Calculate the orientation of the keypoints
points_size=size(list_points,1);
for i=1:points_size     %Look in the keypoints
    octave=list_points(i,3);    %octave of the interest point
    level=(list_points(i,4))+updown;    %level of the interest point
    x1=list_points(i,1);    % x position of the interest point
    y1=list_points(i,2);    % y position of the interest point
    sigma=sigmavalue*k1^(level-1);  %calculate the sigma in the corresponding
scale
    list_points(i,5)=sigma;
    mag=cell2mat(magnitude(level-updown,octave));   %magnitude of the gradient
    ori=cell2mat(orientation(level-updown,octave)); %direction of the gradient
    [weights,n]=gausswindow(1.5*sigma); %create weigths around the interest point
    hist=zeros(36,1);   %create an empty histogram

    [xmax,ymax]=size(mag);
```

Matlab Implementation (Calculate the KP-dominant orientation)

```
a11=x1-n; % Positioning in the KP neighbors
a12=x1+n;

a21=y1-n;
a22=y1+n;

a31=1;
a32=2*n+1;

a41=1;
a42=2*n+1;
if((x1-n)<1) % Left of the minimum size
    a11=1;
   a31=n-x1+2;
end

if((y1-n)<1)
    a21=1;
    a41=n-y1+2;
end

if((x1+n)>xmax)  % Right of the maximum size
    a12=xmax;
    a32=(2*n+1)-(x1+n-xmax);
end

if((y1+n)>ymax)
    a22=ymax;
    a42=(2*n+1)-(y1+n-ymax);
end
```

Matlab Implementation (Calculate the KP-dominant orientation)

```
w=mag(a11:a12,a21:a22).*weights(a31:a32,a41:a42); %weight of each pixel in the bin, after gaussian
                                                    weighting and magnitude assignation.
    ori=ori(a11:a12,a21:a22);

    ori=ceil((ori+pi)*18/pi);   %obtain the direction separated in bins of each one of the points near to
                                  the keypoint

    [orix,oriy]=size(ori);

    %calculate histogram

    for i1=1:orix
        for j1=1:oriy
            hist(ori(i1,j1))=hist(ori(i1,j1))+w(i1,j1);
        end
    end
    [maxval,bin]=max(hist);
    list_points(i,6)=bin;
    lista=find(hist>(0.8*maxval));
    more=size(lista,1);
    if(more>1)
        for i3=1:more
            if(lista(i3) ~= bin)
                new_ones=[new_ones;  [x1 y1 octave list_points(i,4) list_points(i,5) lista(i3)]];
            end
        end
    end


end

sift_desc=[list_points];
```

ShowKeys Function

```
% showkeys(image, locs)
%
% This function displays an image with SIFT keypoints overlayed.
%   Input parameters:
%     image: the file name for the image (grayscale)
%     locs: matrix in which each row gives a keypoint location (row,
%           column, scale, orientation)

function showkeys(image, locs)

disp('Drawing SIFT keypoints ...');

% Draw image with keypoints
figure('Position', [50 50 size(image,2) size(image,1)]);
colormap('gray');
imagesc(image);
hold on;
imsize = size(image);
for i = 1: size(locs,1)
    % Draw an arrow, each line transformed according to keypoint parameters.
    TransformLine(imsize, locs(i,:), 0.0, 0.0, 1.0, 0.0);
    TransformLine(imsize, locs(i,:), 0.85, 0.1, 1.0, 0.0);
    TransformLine(imsize, locs(i,:), 0.85, -0.1, 1.0, 0.0);
end
hold off;
```

ShowKeys Function

```
% ------ Subroutine: TransformLine -------
% Draw the given line in the image, but first translate, rotate, and
% scale according to the keypoint parameters.
%
% Parameters:
%   Arrays:
%    imsize = [rows columns] of image
%    keypoint = [subpixel_row subpixel_column scale orientation]
%
%   Scalars:
%    x1, y1; begining of vector
%    x2, y2; ending of vector
function TransformLine(imsize, keypoint, x1, y1, x2, y2)

% The scaling of the unit length arrow is set to approximately the radius
%   of the region used to compute the keypoint descriptor.
len = 6 * keypoint(3);

% Rotate the keypoints by 'ori' = keypoint(4)
s = sin(keypoint(4));
c = cos(keypoint(4));

% Apply transform
r1 = keypoint(1) - len * (c * y1 + s * x1);
c1 = keypoint(2) + len * (- s * y1 + c * x1);
r2 = keypoint(1) - len * (c * y2 + s * x2);
c2 = keypoint(2) + len * (- s * y2 + c * x2);

line([c1 c2], [r1 r2], 'Color', 'c');
```

Matlab Implementation: Plot KP-Features

```
%%%PLOT FEATURES


dire=((list_points(:,6)/18))*pi + pi/2;

mag=(list_points(:,3))+list_points(:,4);

locs=[y3 x3 mag dire];

showkeys(image, locs);


end
```

Testing the detector

- i=imread('groceries_gray.jpg');
- Sift_Descriptor=sift(i,3,5,1.1);

Vl_feat

- The VLFeat open source library implements popular computer vision algorithms including SIFT, MSER, k-means, hierarchical k-means, agglomerative information bottleneck, and quick shift. It is written in C for efficiency and compatibility, with interfaces in MATLAB for ease of use, and detailed documentation throughout. It supports Windows, Mac OS X, and Linux

Vl_feat

- Download vl_feat from http://www.vlfeat.org/
- run('VLFEATROOT/toolbox/vl_setup')

- Permanent setup
  – To permanently add VLFeat to your MATLAB environment, add this line to your startup.m file:
  – run('VLFEATROOT/toolbox/vl_setup')

Extracting frames and descriptors

```
pfx = fullfile(vl_root,'data', 'roofs1.jpg') ;
I = imread(pfx) ;
image(I) ;
I = single(rgb2gray(I)) ;
[f,d] = vl_sift(I) ;
perm = randperm(size(f,2)) ;
sel = perm(1:50) ;
h1 = vl_plotframe(f(:,sel)) ;
h2 = vl_plotframe(f(:,sel)) ;
set(h1,'color','k','linewidth',3) ;
set(h2,'color','y','linewidth',2) ;
h3 = vl_plotsiftdescriptor(d(:,sel),f(:,sel)) ;
set(h3,'color','g')
```

Basic Matching

```
pfx1 = fullfile(vl_root,'data', 'roofs1.jpg') ;
Ia = imread(pfx1) ;
pfx2 = fullfile(vl_root,'data', 'roofs2.jpg') ;
Ib = imread(pfx2) ;
[fa, da] = vl_sift(Ia) ;
[fb, db] = vl_sift(Ib) ;
[matches, scores] = vl_ubcmatch(da, db) ;
```

Visualization

```
m1= fa (1:2,matches(1,:));
m2=fb(1:2,matches(2,:));
m2(1,:)=
m2(1,:)+640*ones(1,size(m2,2));
X=[m1(1,:);m2(1,:)];
Y=[m1(2,:);m2(2,:)];
imshow(c);
hold on;
line(X,Y)
vl_plotframe(aframe(:,matches(1,:)));
vl_plotframe(m2);
```

Custom frames

- The MATLAB command vl_sift (and the command line utility) can bypass the detector and compute the descriptor on custom frames using the Frames option.
- For instance, we can compute the descriptor of a SIFT frame centered at position (100,100), of scale 10 and orientation -pi/8 by

```
fc = [100;100;10;-pi/8] ;
[f,d] = vl_sift(I,'frames',fc) ;
fc = [100;100;10;0] ;
[f,d] = vl_sift(I,'frames',fc,'orientations') ;
```