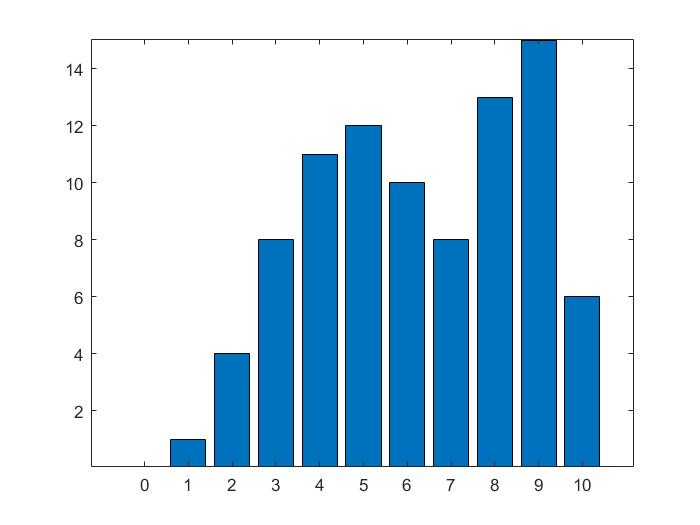
## Problem 1

1. **Consider the filter f=[1, 2, 1] and the 1D image I=[0,1,2,3,3,3,1,3,6]. What is the results of f\*I? Pad the image with zeros at the boundaries if necessary. (10 points)**

Results of f\*I is [0, 1, 4, 8, 11, 12, 10, 8, 13, 15, 6]



Code:

f = [1 2 1];

g = [0 1 2 3 3 3 1 3 6 ];

result = conv(f,g);

x = 0:10;

bar(x,result);

1. **Name two specific ways in which one could reduce the amount of fine, detailed edges that are detected with the Canny edge detector. (10 points)**
2. Choose the high threshold
3. And heavy smoothing
4. **Hybrid images. In this problem you will create hybrid images as described in [1].** Take two images, A and B, that you’ll want to have blend from one to the other. Try to make the objects in the two images occupy more or less the same region. Construct a hybrid image from A (to be seen close-up) and B (to be seen far away) as follows: out = blur(B) + (A-blur(A)) Where blur is a function that low-pass filters the image. You can write your own blur function, or use the upBlur and blurDn functions supplied in [2] (which go up and down. Gaussian pyramid levels). You will want to blur by more than just one Gaussian pyramid level. How does the blurring level affect your perception of the results?



**Code：**

clear;

image1 = "pb3\_1.jpg";

image2 = "pb3\_2.jpg";

img1\_ = imread(image1);

img2\_ = imread(image2);

img1 = double(rgb2gray(img1\_));

img2 = double(rgb2gray(img2\_));

%[row,col] = size(graycat);

sigma = 10;

N = fix(3\*sigma);

N\_row = 2\*N+1;

gausFilter = fspecial('gaussian',[N\_row N\_row],sigma);

blur1 = filter2(gausFilter,img1);

blur2 = filter2(gausFilter,img2);

out = blur1+(img2-blur2);

imshow(out,[]);

**Result:**



Look closely, it is a bear; At a distance, it is crayon shin-chan.

**4. Read the book chapter chapter-local-image-feature-David.pdf (find it in canvas) andcomplete the following programing exercises. (40 points)**

(1) Build a Harris corner detector; for each corner, estimate scale and orientation.

Now test how well your list of neighborhoods behaves under rotation, translation, and scale of the image. You can do this by a simple exercise in matching. For each test image, prepare a rotated, translated, and scaled version of that image. Now you know where each neighborhood should appear in the new version of the image — check how often something of the right size and orientation appears in the right place. You should find that rotation and translation cause no significant problems, but large scale changes can be an issue. (20 points)

(2) Use DoG detector in SIFT to detect the interesting points for the same test image, and compare the results with Harris corner detector on the variances of rotation, translation and scaling. (20 points)

**(1)**

Code:

clear;

%Harris corner detector

image = imread("pb3\_1.jpg");

img = double(rgb2gray(image));

[row,col]=size(img);

%gaussian filter parameter

sigma = 1.6;

%N = 3; %kernel size is (2N+1)\*(2N+1)

N\_row = fix(6\*sigma);

gausFilter = fspecial('gaussian',[N\_row N\_row],sigma);

%%applying sobel edge detector in the horizontal direction

fx = [-1 0 1;-2 0 2;-1 0 1];

Ix = filter2(fx,img);

% applying sobel edge detector in the vertical direction

fy = fx';

Iy = filter2(fy,img);

Ix2 = Ix.^2;

Iy2 = Iy.^2;

Ixy = Ix.\*Iy;

% optionaly, gaussion filter

Ix2 = filter2(gausFilter,Ix2);

Iy2 = filter2(gausFilter,Iy2);

Ixy = filter2(gausFilter,Ixy);

R = zeros(row,col);

for i = 1:row

for j = 1:col

M = [Ix2(i,j) Ixy(i,j);Ixy(i,j) Iy2(i,j)];

R(i,j) = det(M)-0.05\*(trace(M))^2;

end

end

allmax = max(max(R));

R = R/allmax;

threshold = 0.1;

R(R<threshold)=0;

points = imregionalmax(R);

%points = nlfilter(R, [3 3], @(x) all(x(5)> x([1:4 6:9])) );

[posr, posc] = find(points == 1);

imshow(img,[]);

hold on;

plot(posc,posr,'r.');

hold on;

[~,Gdir] = imgradient(img);

ptx = posr(10);

pty = posc(10);

point = [ptx,pty];

%[best\_scale,estimate\_response,main\_orient]= estimate\_scale\_and\_orientation(img,Gdir,point,1);

%corner point struct

for i = 1:length(posr)

point = [posr(i),posc(i)];

Pt{i}= estimate\_scale\_and\_orientation(img,Gdir,point,1);

x = Pt{i}.x-Pt{i}.scale;

y = Pt{i}.y-Pt{i}.scale;

w = 2\*Pt{i}.scale;

h = 2\*Pt{i}.scale;

orientx = Pt{i}.x;

orienty = Pt{i}.y;

u = 100\*cos(Pt{i}.orient/180\*pi);

v = 100\*sin(Pt{i}.orient/180\*pi);

pos = [y x w h]; %%ÒòÎªÍ¼Æ¬ÊÇÐý×ªÁËµÄ£¬ËùÒÔxÎ»ÖÃºÍyÎ»ÖÃÒª×ªÒ»ÏÂ

rectangle('Position',pos,'Curvature',[1 1]);

hold on;

quiver(orienty,orientx,v,u);

hold on;

end

function Pt = estimate\_scale\_and\_orientation(img,Gdir,point,dpixel)

[X,Y] = size(img);

ptx = point(1);

pty = point(2);

max\_radius = min(min(X-ptx,ptx-1),min(Y-pty,pty-1));

scale\_range = 1:dpixel:max\_radius;

sigma\_range = 0.3\*(scale\_range-1)+0.8;

estimate\_response = zeros(1,length(scale\_range));

for i = 1:length(scale\_range)

patch = img(ptx-scale\_range(i):ptx+scale\_range(i),pty-scale\_range(i):pty+scale\_range(i));

LoGfilter = fspecial('log',[2\*scale\_range(i)+1,2\*scale\_range(i)+1],sigma\_range(i));

estimate\_response(i) = sum(sum(LoGfilter.\*patch));

end

%argmax, obtain the best scale

[ ~, idx ] = max(estimate\_response);

best\_scale = scale\_range(idx);

%calculate the orientation

orient\_patch = Gdir(ptx-best\_scale:ptx+best\_scale,pty-best\_scale:pty+best\_scale);

all\_orient = Obtain\_orient(orient\_patch); %locate the orientation to certain orientation part

%statistic of the orientation

hist = tabulate(all\_orient(:));

[~,hist\_idx]=max(hist(:,2));

main\_orient = hist(hist\_idx,1);

Pt.x = ptx;

Pt.y = pty;

Pt.scale = best\_scale;

Pt.orient = main\_orient;

end

function dirLoc = Obtain\_orient(dir)

if dir>0

r = fix((dir+5)/10);

dirLoc = r\*10;

else

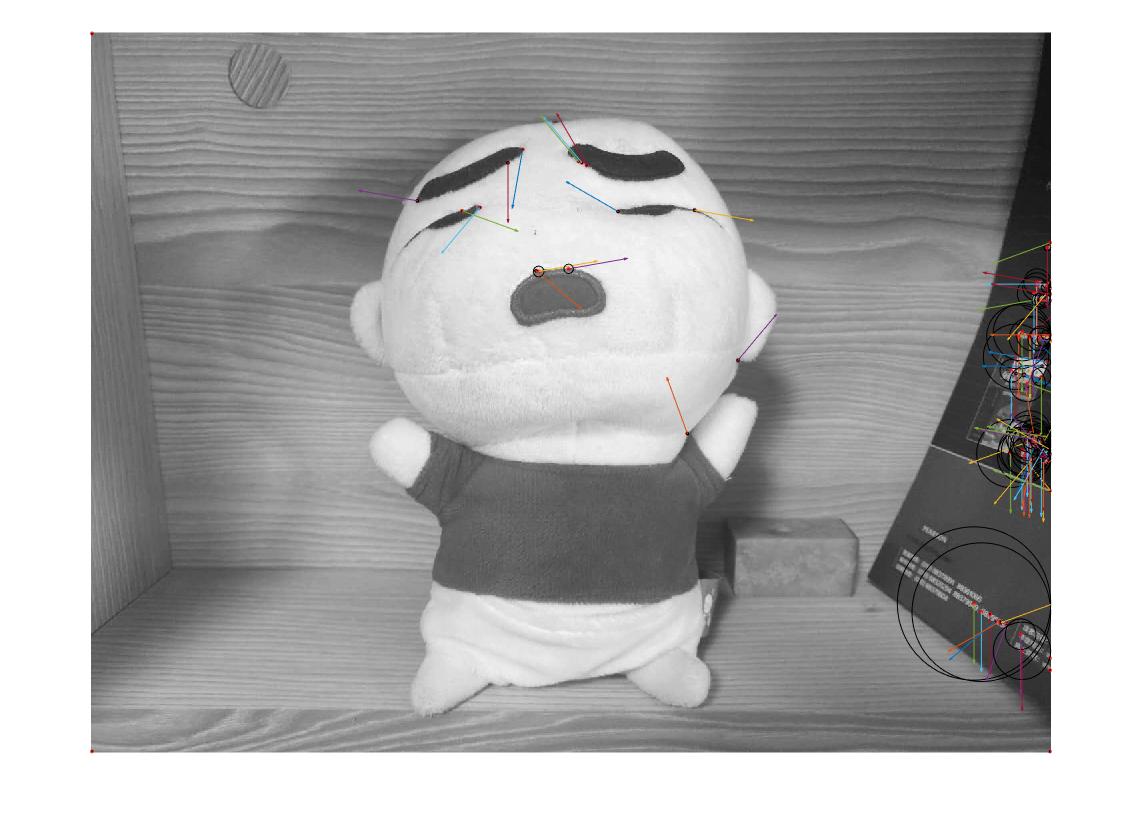
r = fix((dir-5)/10);

dirLoc = r\*10;

end

end

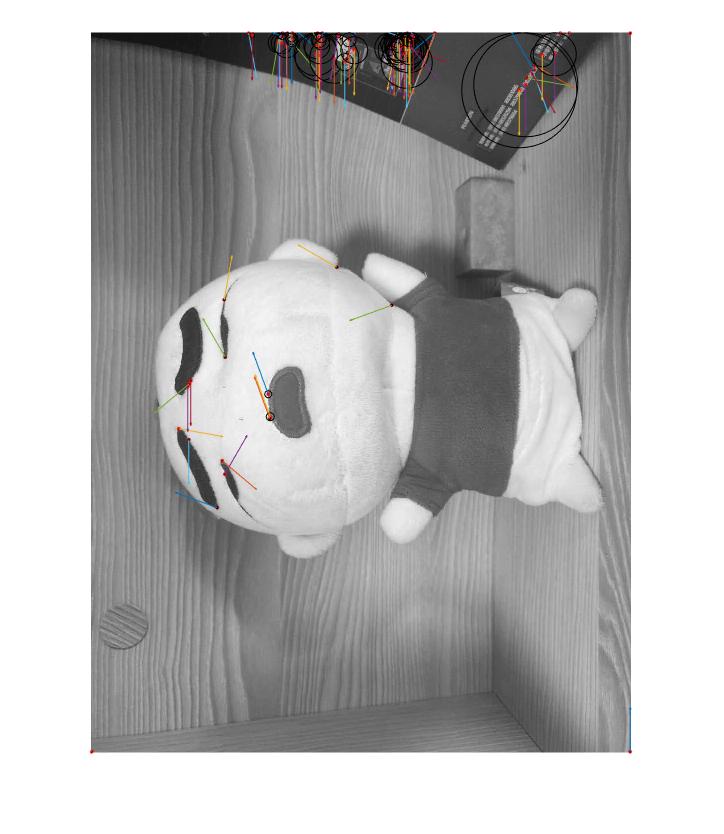
Result:



The circle represents scale, the radius of the circle is scale.

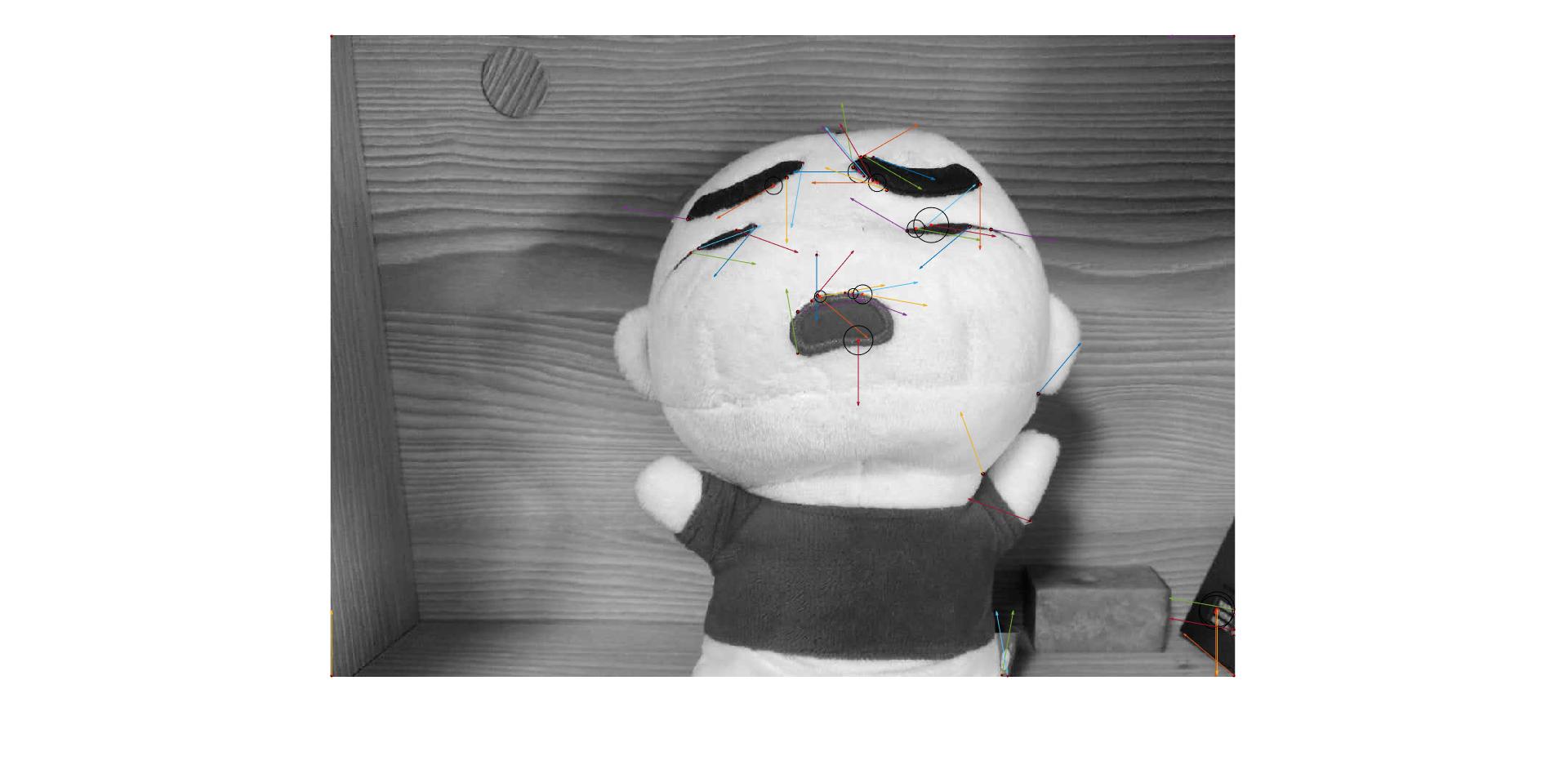
The arrow means the main orientation of the corner.

The rotated image: there is some orientation change when rotating the image, but not significant problem.



The translation image:

Since I use the 0.01\*max value as the harris response threshold, and the translated image may change the point of max value, so we have more corner points. But it is not the significant problem.



Scale of the image:

By comparing this image with original image, we can find that the scale of the image have relatively large influence at scale (size or sigma).



**(2)**

**Code:**

clear;

image1 = "cat.png";

I = imread(image1);

exterma = key\_points(I);

function exterma=key\_points(I)

%I=imread('Image');

I=double(rgb2gray(I));

I=I/max(max(I)); % image should be in [0 1]

[M,N] = size(I) ;

S=3 ;

omin=-1 ; % first octave -1 mmeans I should be doublsized for first octave

O=floor(log2(min(M,N)))-omin-4 ; % Up to 16x16 images

sigma0=1.6\*2^(1/S) ;

sigman=0.5

thresh = 0.006;

r = 10 ;

GS = gaussianss(I,O,S,omin,-1,S+1,sigma0) ;

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%calculate DOG images

for o=1:GS.O %all the octaves

[M,N,SS] = size(GS.octave{o}) ;

DOG.octave{o} = zeros(M,N,SS-1) ;

for s=1:SS-1

DOG.octave{o}(:,:,s) = ...

GS.octave{o}(:,:,s+1) - GS.octave{o}(:,:,s) ;

end

end

%finding key points

exterma=zeros(2,1);

for o=1:GS.O

for s=2:SS-2

sig=1.6\*2^(o-1)\*(2^(1/S))^s;

current\_DOG=DOG.octave{o}(:,:,s);

down\_DOG=DOG.octave{o}(:,:,s-1);

up\_DOG=DOG.octave{o}(:,:,s+1);

extr = search\_exterm(up\_DOG,down\_DOG,current\_DOG ) ;%find exremum

if extr(1,1)

extr=localize\_eliminate(extr,up\_DOG,down\_DOG,current\_DOG ,thresh,r);

if extr(1,1)

extr=2^(o-1+GS.omin) \*extr; %stor key points

exterma=[exterma extr];

end

end

end

end

imshow(I,[])

hold on

plot(exterma(2,:),exterma(1,:),'r+','LineWidth',2)

hold on

exterma = fix(exterma);

[~,Gdir] = imgradient(I);

for i = 2:length(exterma)

point = [exterma(1,i),exterma(2,i)];

Pt{i}= estimate\_scale\_and\_orientation(I,Gdir,point,1);

x = Pt{i}.x-Pt{i}.scale;

y = Pt{i}.y-Pt{i}.scale;

w = 2\*Pt{i}.scale;

h = 2\*Pt{i}.scale;

orientx = Pt{i}.x;

orienty = Pt{i}.y;

u = 100\*cos(Pt{i}.orient/180\*pi);

v = 100\*sin(Pt{i}.orient/180\*pi);

pos = [y x w h]; %%ÒòÎªÍ¼Æ¬ÊÇÐý×ªÁËµÄ£¬ËùÒÔxÎ»ÖÃºÍyÎ»ÖÃÒª×ªÒ»ÏÂ

rectangle('Position',pos,'Curvature',[1 1]);

hold on;

quiver(orienty,orientx,v,u);

hold on;

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function SS = gaussianss(I,O,S,omin,smin,smax,sigma0)

%smax--> maximum scale(here it is 4)

%smin--> minimum scale(here is -1...image shold be double sized)

%omin--> first octave(here is -1)

%1.6 as sigma0 is considered for omin=-1

% Scale multiplicative step

k = 2^(1/S) ;

dsigma0 = sigma0 \* sqrt(1 - 1/k^2) ; % Scale step factor

sigman = 0.5 ; % Nominal smoothing of the image

% Scale space structure

SS.O = O ;

SS.S = S ;

SS.sigma0 = sigma0 ;

SS.omin = omin ;

SS.smin = smin ;

SS.smax = smax ;

if omin < 0

for o=1:-omin

I = doubleSize(I) ;

end

elseif omin > 0

for o=1:omin

I = halveSize(I) ;

end

end

[M,N] = size(I) ;

% Index offset

so = -smin+1 ;

if(sigma0 \* 2^omin \* k^smin < sigman)

warning('The nominal smoothing exceeds the lowest level of the scale space.') ;

end

SS.octave{1} = zeros(M,N,smax-smin+1) ;% we have 6 scale in each octave

SS.octave{1}(:,:,1) = gauss\_filter(I,sqrt((sigma0\*k^smin)^2 ...

- (sigman/2^omin)^2));

for s=smin+1:smax

dsigma = k^s \* dsigma0 ;% smooth Image in prevous scale and just use dsigma

SS.octave{1}(:,:,s +so) =gauss\_filter...

(squeeze(SS.octave{1}(:,:,s-1 +so)), dsigma);

%HOSSEIN imsmooth(squeeze(SS.octave{1}(:,:,s-1 +so)), dsigma ) ;

end

% --------------------------------------------------------------------

% Other octaves

% --------------------------------------------------------------------

for o=2:O

sbest = min(smin + S, smax) ;

TMP = halveSize(squeeze(SS.octave{o-1}(:,:,sbest+so))) ;

target\_sigma = sigma0 \* k^smin ;

prev\_sigma = sigma0 \* k^(sbest - S) ;

if (target\_sigma > prev\_sigma)

TMP =gauss\_filter(TMP, sqrt(target\_sigma^2 - prev\_sigma^2));

end

[M,N] = size(TMP) ;

SS.octave{o} = zeros(M,N,smax-smin+1) ;

SS.octave{o}(:,:,1) = TMP ;

for s=smin+1:smax

% The other levels are determined as above for the first octave.

dsigma = k^s \* dsigma0 ;

SS.octave{o}(:,:,s +so) =gauss\_filter(squeeze(SS.octave{o}...

(:,:,s-1 +so)), dsigma);

end

end

end

% -------------------------------------------------------------------------

% Auxiliary functions

% -------------------------------------------------------------------------

function J = doubleSize(I)

[M,N]=size(I) ;

J = zeros(2\*M,2\*N) ;

J(1:2:end,1:2:end) = I ;

J(2:2:end-1,2:2:end-1) = ...

0.25\*I(1:end-1,1:end-1) + ...

0.25\*I(2:end,1:end-1) + ...

0.25\*I(1:end-1,2:end) + ...

0.25\*I(2:end,2:end) ;

J(2:2:end-1,1:2:end) = ...

0.5\*I(1:end-1,:) + ...

0.5\*I(2:end,:) ;

J(1:2:end,2:2:end-1) = ...

0.5\*I(:,1:end-1) + ...

0.5\*I(:,2:end) ;

end

function J = halveSize(I)

J=I(1:2:end,1:2:end) ;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function im=gauss\_filter(image,sigma)

G = fspecial('gaussian',[5 5],sigma);

im=imfilter(image,G,'same');

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [points2]=localize\_eliminate(points,up,down,curr,thr,r)

points2=zeros(2,1);

t=1;

for i=1:size(points,2)

x=points(1,i);

y=points(2,i);

fxx= curr(x-1,y)+curr(x+1,y)-2\*curr(x,y); % double derivate in x direction

fyy= curr(x,y-1)+curr(x,y+1)-2\*curr(x,y); % double derivate in y direction

fsigmasigma=up(x,y)+down(x,y)-2\*curr(x,y); % double derivate in sigma direction

fxsigma=((up(x+1,y)-down(x+1,y))-(up(x-1,y)-down(x-1,y)))/4;%derivate in x and sigma direction

fysigma=((up(x,y+1)-down(x,y+1))-(up(x,y-1)-down(x,y-1)))/4;%derivate in y and sigma direction

fxy= curr(x-1,y-1)+curr(x+1,y+1)-curr(x-1,y+1)-curr(x+1,y-1); %derivate inx and y direction

fx=curr(x,y)-curr(x-1,y);%derivate in x direction

fy=curr(x,y)-curr(x,y-1);%derivate in y direction

fsigma=(up(x,y)-down(x,y))/2;%derivate in sigma direction

%localization using Teilor seri

A=[fsigmasigma fxsigma fysigma;fxsigma fxx fxy;fysigma fxy fyy];

X=-inv(A)\*([fsigma fx fy]');

x\_hat=X(2);

y\_hat=X(3);

if abs(x\_hat)<4 && abs(y\_hat)<4 %ignor the ofsets > 4

px=round(x+x\_hat);

py=round(y+y\_hat);

else

px=x;

py=y;

%[px py]

end

D\_hat=curr(px,py)+([fsigma fx fy]\*X)/2;

if abs(D\_hat)>thr%% filter some low contrast points

if (fxx+fyy)^2/(fxx\*fyy-fxy^2)<(r+1)^2/r % remove edge points

points2(1,t)=px;points2(2,t)=py;

t=t+1;

end

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function indx=search\_exterm(up,down,im)

[m n]=size(im);

t=1;

thr=.004;

indx=[0;0];

for i=2:m-1

for j=2:n-1

if im(i,j)> thr

window(1:3,1:3)=down(i-1:i+1,j-1:j+1);

window(4:6,1:3)=im(i-1:i+1,j-1:j+1);

window(7:9,1:3)=up(i-1:i+1,j-1:j+1);

window(5,2)=-100;

if im(i,j)>max(max(window))

indx(:,t)=[i j]';

t=t+1;

end

end

if im(i,j)<-thr

window(1:3,1:3)=down(i-1:i+1,j-1:j+1);

window(4:6,1:3)=im(i-1:i+1,j-1:j+1);

window(7:9,1:3)=up(i-1:i+1,j-1:j+1);

window(5,2)=100;

if im(i,j)<min(min(window))

indx(:,t)=[i j]';

t=t+1;

end

end

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function Pt = estimate\_scale\_and\_orientation(img,Gdir,point,dpixel)

[X,Y] = size(img);

ptx = point(1);

pty = point(2);

max\_radius = min(min(X-ptx,ptx-1),min(Y-pty,pty-1));

scale\_range = 1:dpixel:max\_radius;

sigma\_range = 0.3\*(scale\_range-1)+0.8;

estimate\_response = zeros(1,length(scale\_range));

for i = 1:length(scale\_range)

patch = img(ptx-scale\_range(i):ptx+scale\_range(i),pty-scale\_range(i):pty+scale\_range(i));

LoGfilter = fspecial('log',[2\*scale\_range(i)+1,2\*scale\_range(i)+1],sigma\_range(i));

estimate\_response(i) = sum(sum(LoGfilter.\*patch));

end

%argmax, obtain the best scale

[ ~, idx ] = max(estimate\_response);

best\_scale = scale\_range(idx);

%calculate the orientation

orient\_patch = Gdir(ptx-best\_scale:ptx+best\_scale,pty-best\_scale:pty+best\_scale);

all\_orient = Obtain\_orient(orient\_patch); %locate the orientation to certain orientation part

%statistic of the orientation

hist = tabulate(all\_orient(:));

[~,hist\_idx]=max(hist(:,2));

main\_orient = hist(hist\_idx,1);

Pt.x = ptx;

Pt.y = pty;

Pt.scale = best\_scale;

Pt.orient = main\_orient;

end

function dirLoc = Obtain\_orient(dir)

if dir>0

r = fix((dir+5)/10);

dirLoc = r\*10;

else

r = fix((dir-5)/10);

dirLoc = r\*10;

end

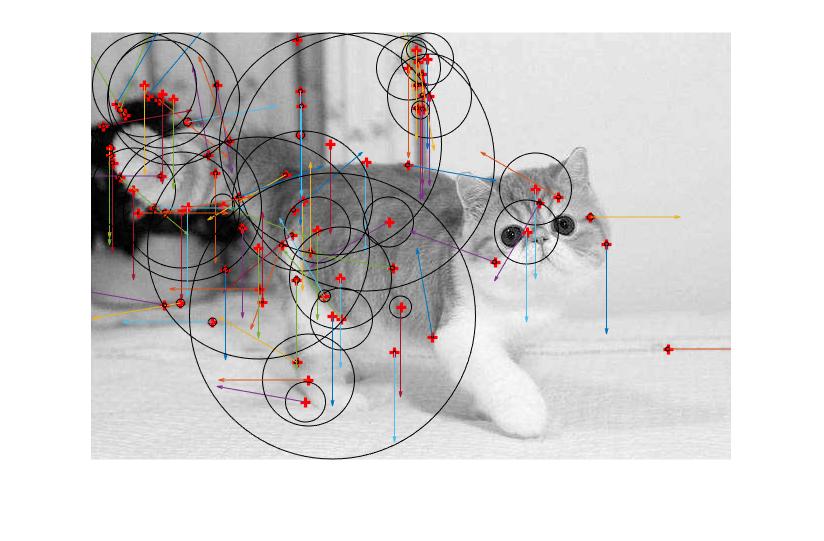
end

**Result:**

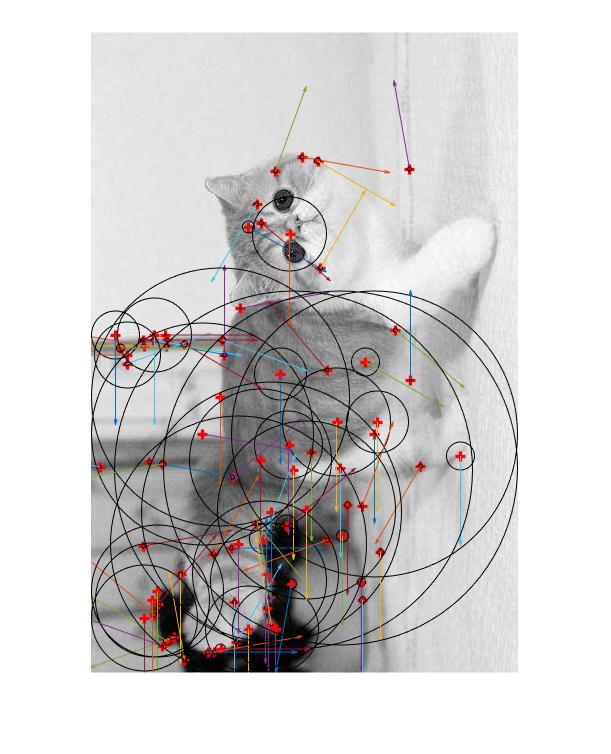
The original picture:



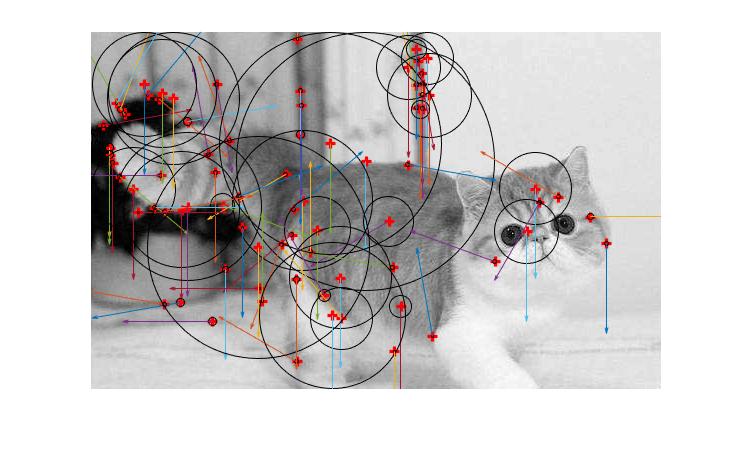
Result with sift points:



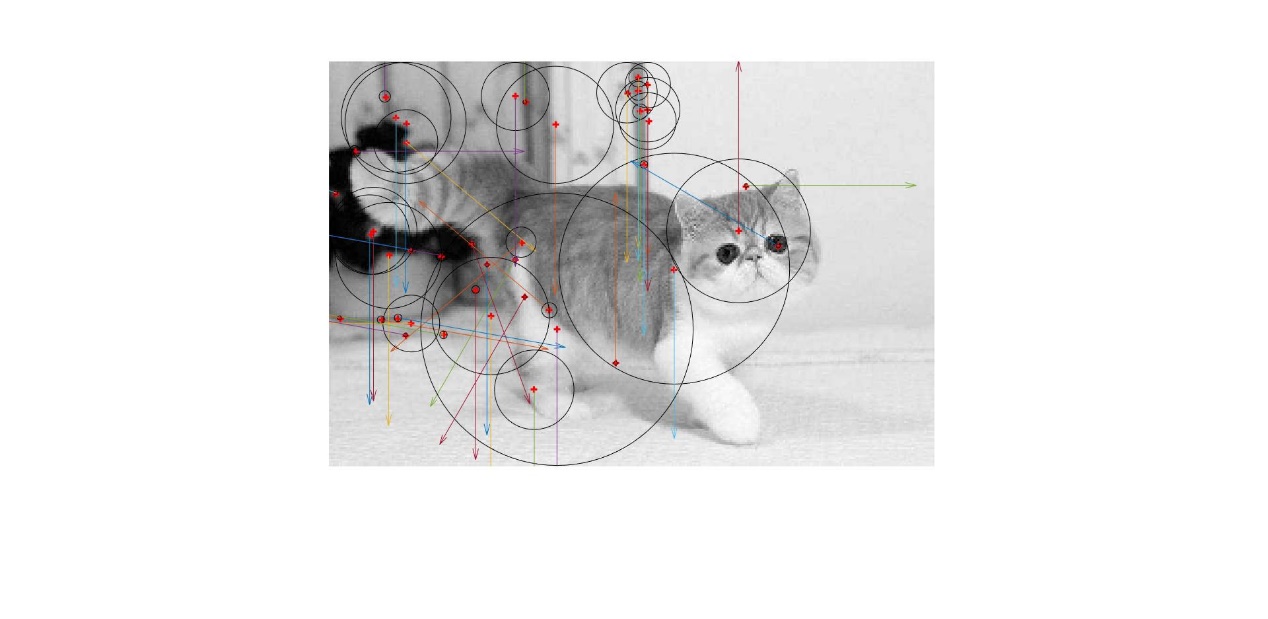
Rotated picture:



Translated image:



Scaled image:



The SIFT have relatively small variance with translation, scale and rotation.

Though the scaled image also change the scale (sigma or size).