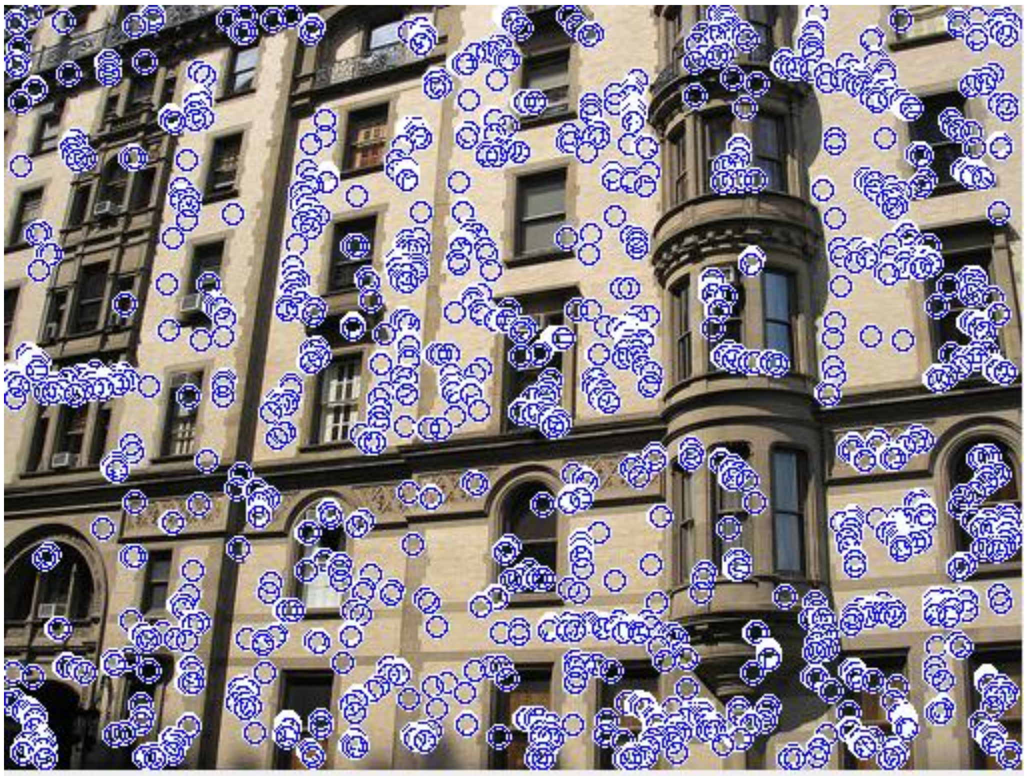
CSC420 Assignment 2

Chuhan Chen 1002202742

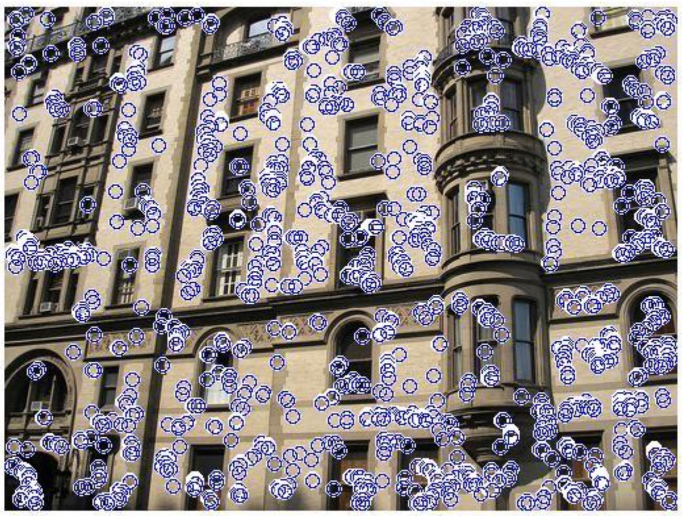
1. a) Below is the produced image, alpha = 0.249.

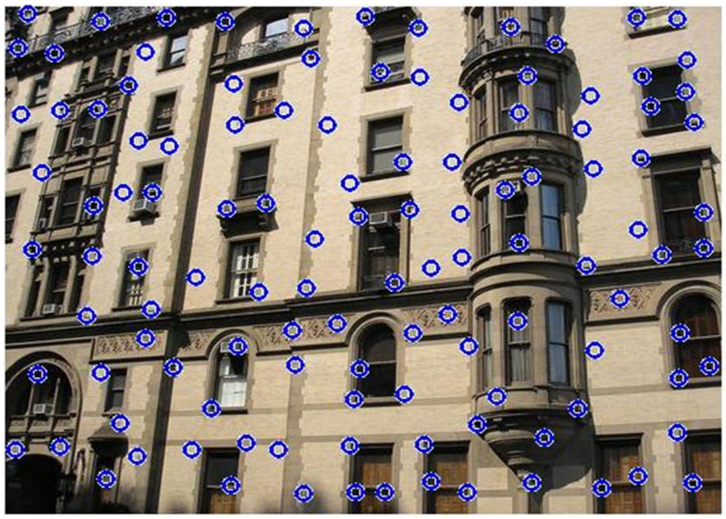


```
function [is_corner] = Harris_corner_detect(I)
    [Ix,Iy] = gradient(I);
    M = zeros(2,2);
    sigma= 15;
    alpha = 0.249;

    for x = 1:size(I,2)
        for y = 1:size(I,1)
            ix = Ix(y,x);
            iy = Iy(y,x);
            M = M + Gaussian2d(x,y,sigma).*[ix.^2 ix*iy;ix*iy iy.^2];
        end
    end

    if((det(M)-alpha*trace(M).^2)>0)
        is_corner=1;
    else
        is_corner = 0;
    end
```

```
input_image = double(rgb2gray(imread( fullfile( './' , 'building.jpg' ))));
patch_size=11;
extend_range = (patch_size-1)/2;
imshow(imread( fullfile( './' , 'building.jpg' )));
for y = 1:size(input_image,1)
    for x = 1:size(input_image,2)
        if(y-extend_range >= 1 && y+extend_range <= size(input_image,1) && x-extend_range >= 1 && x+extend_range <= size(input_image,2))
            if(Harris_corner_detect(input_image(y-extend_range:y+extend_range,x-extend_range:x+extend_range)))
                drawnow
                viscircles([x y],5, 'color','b','linewidth',1)
            end
        end
    end
end
```

1 b) The left image used a circular filter of radius 10, and the right image used a circular filter of radius 20. When we convolve a larger filter with an image, it is more difficult for the current pixel to be equal to the local maximum of the pixels covered by the filter, thus fewer points are detected as corners. On the other hand, when we convolve an image with a smaller filter, it is easier for a pixel to become the local maximum of under the filter. It detects more points with high R values, but they may not be high enough to be an actual corner, resulting in more outliers.



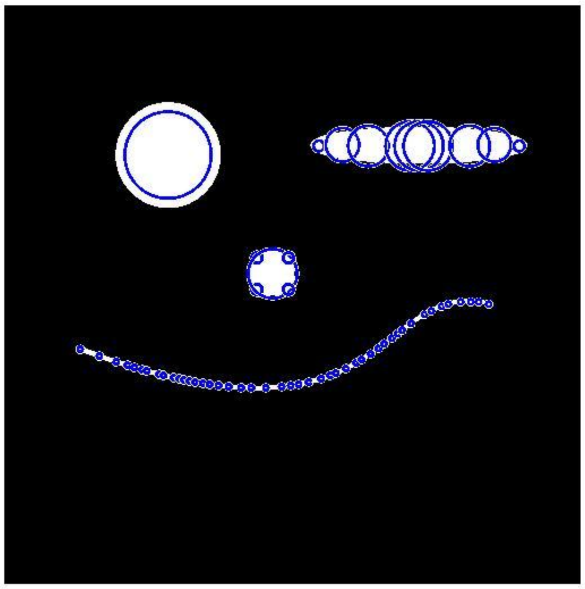
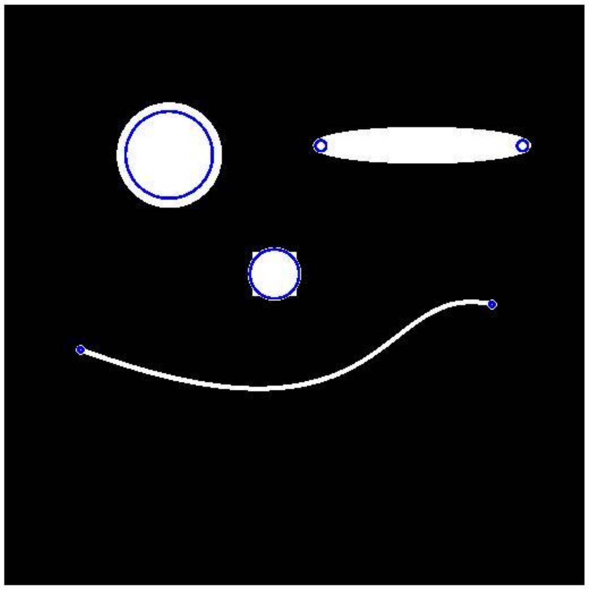


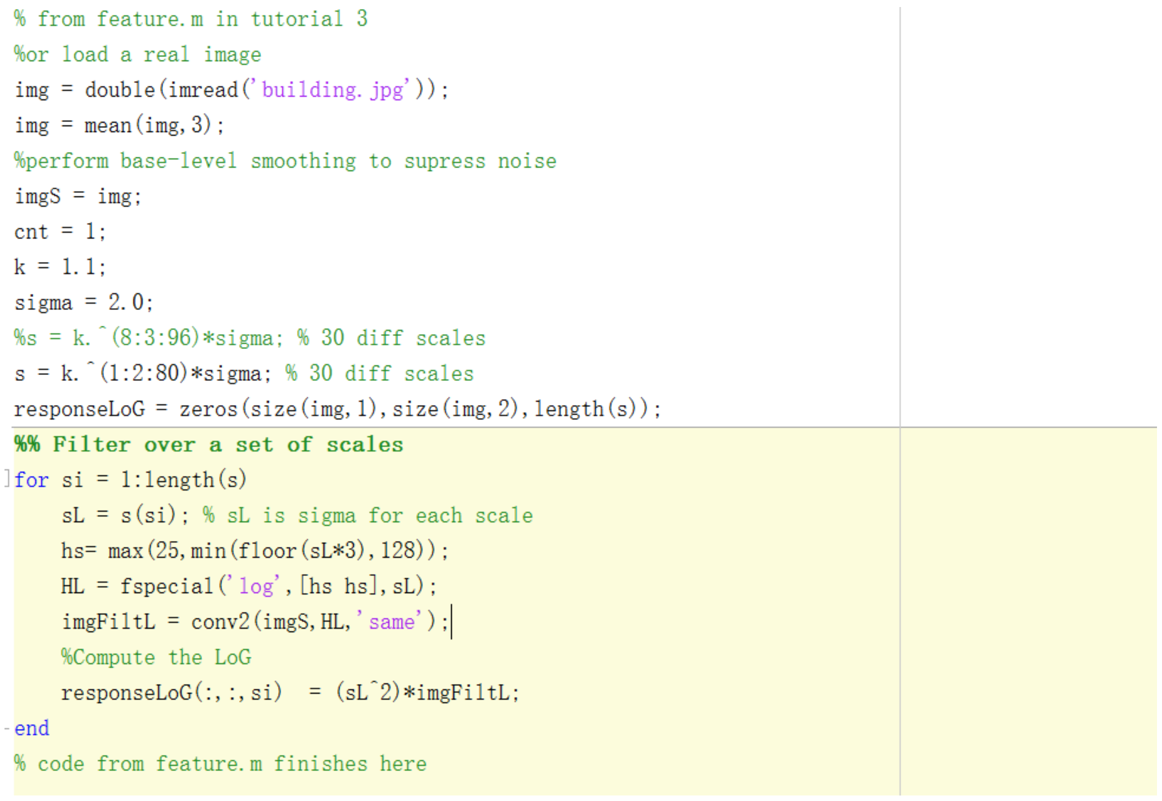
```
function [R] = Harris_corner_detect_returnR(I)
    % pass in an image patch
    [Ix,Iy] = gradient(I);
    M = zeros(2,2);
    sigma= 15;
    alpha = 0.249;

    for x = 1:size(I,2)
        for y = 1:size(I,1)
            ix = Ix(y,x);
            iy = Iy(y,x);
            M = M + Gaussian2d(x,y,sigma).*[ix.^2 ix*iy;ix*iy iy.^2];
        end
    end

    R = det(M)-alpha*trace(M).^2;
```

```
input_image = double(rgb2gray(imread('building.jpg')));

patch_size=11;
extend_range = (patch_size-1)/2;
cornered_points = zeros(size(input_image));
for y = 1:size(input_image,1)
    for x = 1:size(input_image,2)
        if(y-extend_range >= 1 && y+extend_range <= size(input_image,1) && x-extend_range >= 1 && x+extend_range <= size(input_image,2))
            cornered_points(y,x) = Harris_corner_detect_returnR(input_image(y-extend_range:y+extend_range,x-extend_range:x+extend_range));
        end
    end
end
```

```
Radius = 10;
domain = fspecial('disk', Radius);
cornered_points_suppressed = ordfilt2(cornered_points,nnz(domain),domain);

figure;
imshow(imread('building.jpg'));

for y = 1:size(input_image,1)
    for x = 1:size(input_image,2)
        if(cornered_points_suppressed(y,x) > 0 && cornered_points_suppressed(y,x) <= cornered_points(y,x))
                drawnow
                viscircles([x y],5,'color','b');
        end
    end
end
```

1 c) The left image is produced with a positive LoG threshold of 68 and negative threshold of -90 after we find pixels that are extrema in both location and scale. The right image is produced with a positive of 68 and a negative threshold of -105. Higher threshold in magnitude get removes more outliers, yet producing fewer interest points at the same time.

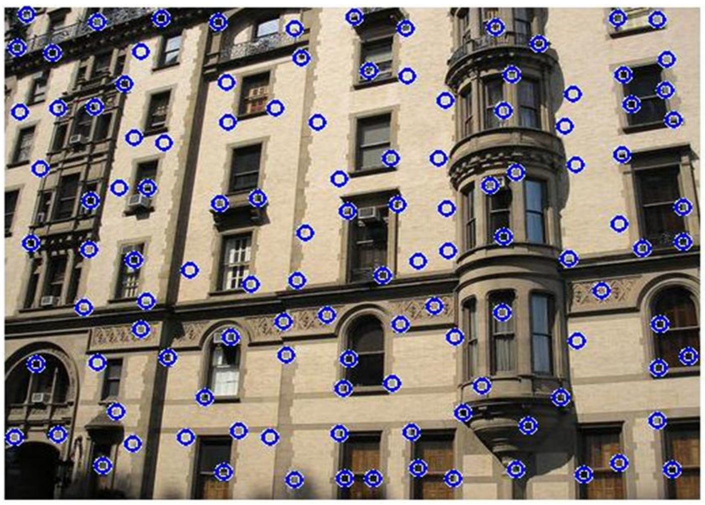
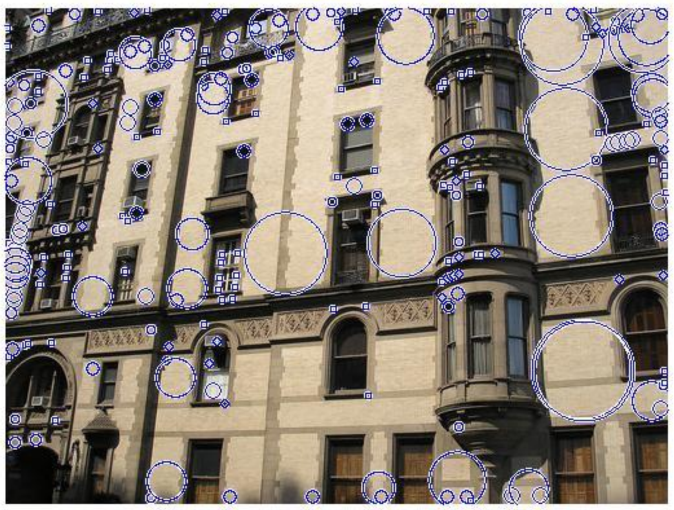


```
% from feature.m in tutorial 3
%or load a real image
img = double(imread('building.jpg'));
img = mean(img,3);
%perform base-level smoothing to supress noise
imgS = img;
cnt = 1;
k = 1.1;
sigma = 2.0;
%s = k.^(8:3:96)*sigma; % 30 diff scales
s = k.^(1:2:80)*sigma; % 30 diff scales
responseLoG = zeros(size(img,1),size(img,2),length(s));
%% Filter over a set of scales
for si = 1:length(s)
    sL = s(si); % sL is sigma for each scale
    hs= max(25,min(floor(sL*3),128));
    HL = fspecial('log',[hs hs],sL);
    imgFiltL = conv2(imgS,HL,'same');
    %Compute the LoG
    responseLoG(:,:,si)  = (sL^2)*imgFiltL;
end
% code from feature.m finishes here
```

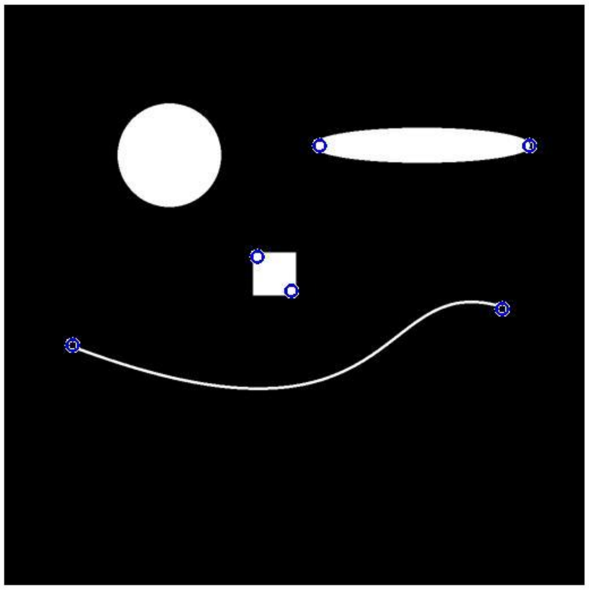
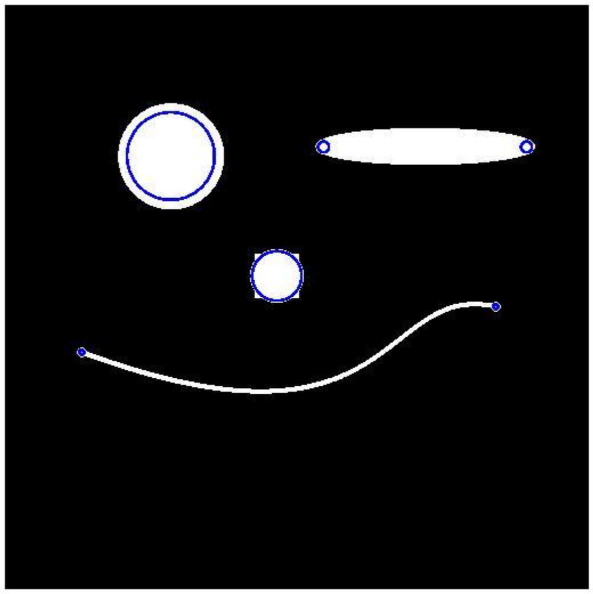
```
imshow('building.jpg');
for x = 1:size(responseLoG,2)
    for y = 1:size(responseLoG,1)
        z_m = find_max(x,y,responseLoG,1);
        if( z_m ~= 0)
                drawnow
                viscircles([x y],s(z_m), 'color','b','linewidth',1);
                key_point = vertcat(key_point,[x, y, z_m]);
        end
    end
end
```

```
function [z] = find_max(x, y, Image_matrix,window_size)
    if (y-window_size < 1 || y+window_size > size(Image_matrix,1) || x-window_size < 1 || x+window_size > size(Image_matrix,2))
        z = 0;
    else
        window = zeros(window_size*2+1,window_size*2+1,size(Image_matrix,3));
        window(:,:,:)  = Image_matrix(y-window_size:y+window_size,x-window_size:x+window_size,:);

        [maxx, i] = max(window(:));
        [minn, i_min] = min(window(:));
        [x_max,y_max,z_max] = ind2sub(size(window),i);
        [x_min,y_min,z_min] = ind2sub(size(window),i_min);
        if(x_max == window_size+1 && y_max == window_size+1 && Image_matrix(y,x,z_max)>70)
            z = z_max;
        elseif(x_min == window_size+1 && y_min == window_size+1 && Image_matrix(y,x,z_min)< -50)
            z = z_min;
        else
            z = 0 ;
         %   result_binary = 0;
        end
    end
```

1 d) 'building.jpg ' using Harris Corner Detection with non-maximum suppression (left) and with SIFT (right):

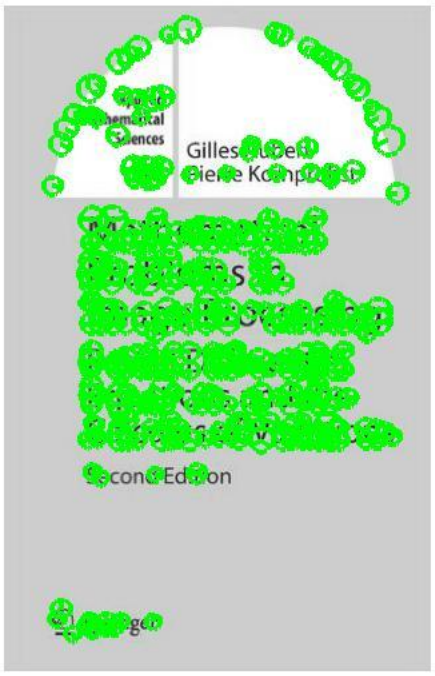
‘synthetic.png’ with using Harris Corner Detection with non-maximum suppression (left) and with SIFT (right):

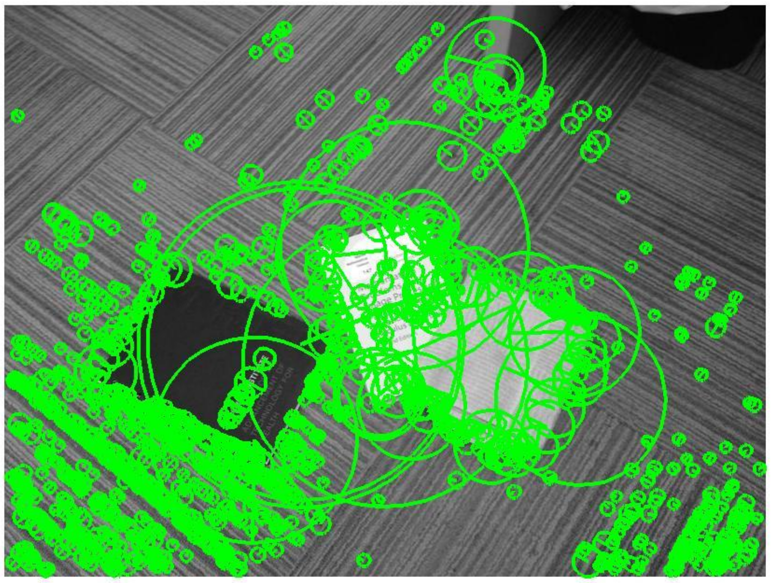



In the both examples, using Harris corner metrics can more accurately detect corners while using SIFT is able to detect interest points that are invariant of scales. For example, in ‘building .jpg’, Harris metric successfully detects most of the corners of windows and doors, while SIFT although not detecting as many corners, detects window surfaces and shapes on the walls of various sizes that become key interest points. In addition, in the ‘synthetic.png’ example, Harris detects corners such as corners of the month, nose and eyes, as these points have sudden change in direction while SIFT detects ‘blobs’ such as the eyes and nose. This is because SIFT employs Laplacian of Gaussian function with different sigma values to detect interest points that have extrema in both location and scale, while Harris corner detection uses gradient to detect direction of changes, which is only in terms of location.

2 a) I used the SIFT library from http://vision.ucla.edu/~vedaldi/assets/sift/binaries/, suggested in sift_demo.m from tutorial 3. Here are the code to produce the following images with the reference of the same file.

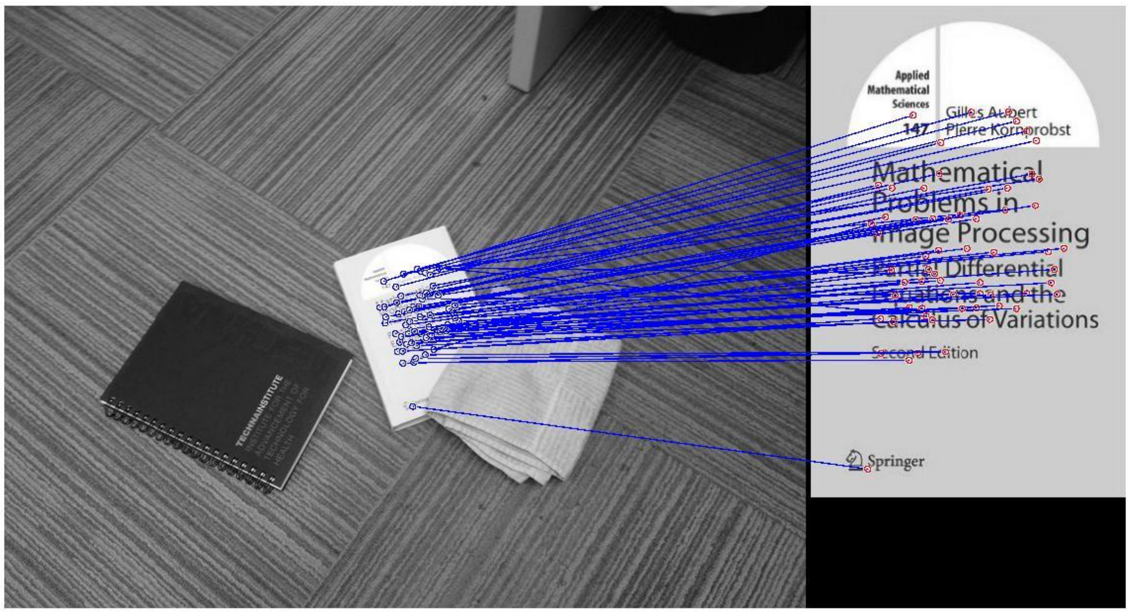
```
img1=double(rgb2gray(imread('./book.jpg')))/255;
img2=double(rgb2gray(imread('./findBook.jpg')))/255;

addpath(genpath('./sift'));
[f1,d1]=sift(img1);
[f2,d2]=sift(img2,'Threshold',0.02);

figure;
imshow(img1);
hold on;
plotsiftframe(f1(:,1:500));
hold off;
figure;
imshow(img2);
hold on;

plotsiftframe(f2(:,1:500));
hold off;
```





2 b)

```
% d2, img2,f2 -- findBook.jpg ; d1, img1,f1 -- book.jpg
match_matrix = zeros(size(d1,2),size(d2,2));
match_matrix(:,:) = pdist2(d1',d2');
max_two = zeros(size(match_matrix,1),2);
indeces_first_max = zeros(size(match_matrix,1),1);

[max_two(:,1),indeces_first_max(:)] = min(match_matrix,[],2);
for i = 1:length(indeces_first_max)
    match_matrix(i,indeces_first_max(i)) = 1000;
end
max_two(:,2) = min(match_matrix,[],2);
ratio = max_two(:,1)./max_two(:,2);

threshold = 0.7;
indeces_matched = (ratio < threshold) & indeces_first_max;

img_to_show = zeros(size(img2,1),size(img2,2)+size(img1,2)+5);
img_to_show(1:size(img2,1),1:size(img2,2)) = img2(:,:);
img_to_show(1:size(img1,1),size(img2,2)+6:size(img2,2)+5+size(img1,2)) = img1(:,:);

imshow(img_to_show);
for i = 1:length(indeces_matched)
    if(indeces_matched(i)~=0)
        lineX = [ f2(1,indeces_first_max(i)),f1(1,i)+816+5 ] ;
        lineY = [  f2(2,indeces_first_max(i)), f1(2,i)] ;
        drawnow
        line( lineX , lineY, 'Color', [0 0 1], 'LineWidth', 1 ) ;
        viscircles([lineX(1),lineY(1)],3, 'color','b','LineWidth', 1);
        viscircles([lineX(2),lineY(2)],3, 'color','r','LineWidth', 1);
    end
end
```

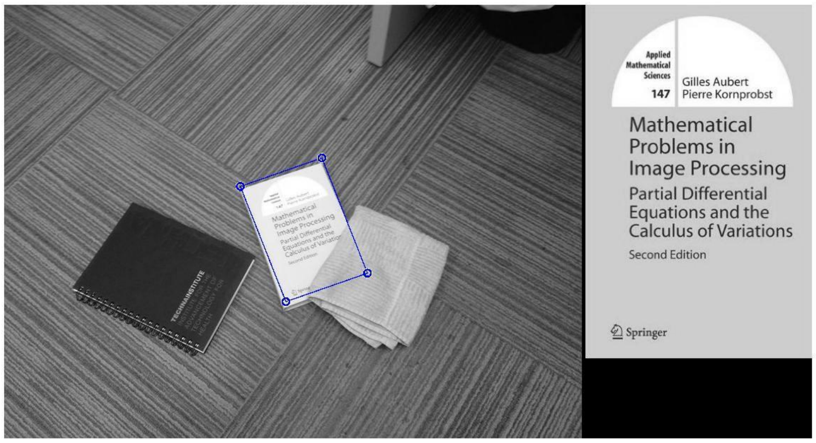

Applied Mathematical Sciences
147
Gilles Aubert
Pierre Kornprobst
Mathematical Problems in Image Processing
Partial Differential Equations and the Calculus of Variations
Second Edition
Springer

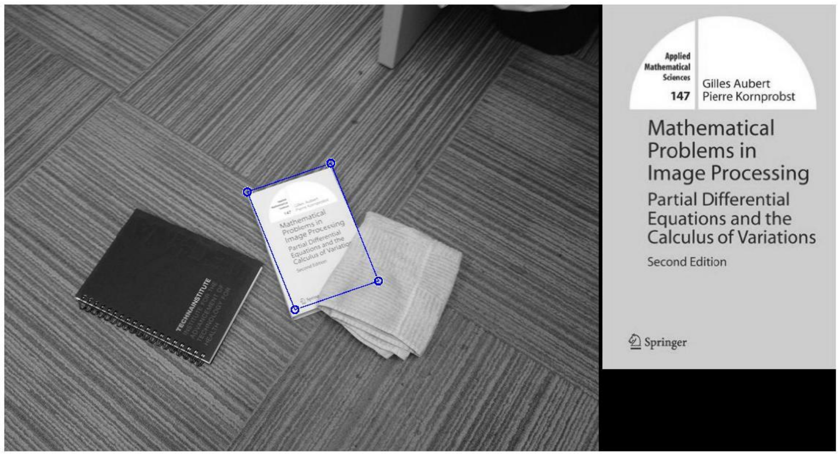
2 c) Here I have included code for both c) and d), affine transform is named as ‘a’, bolded below, results with varying k is demonstrated in 2 d).

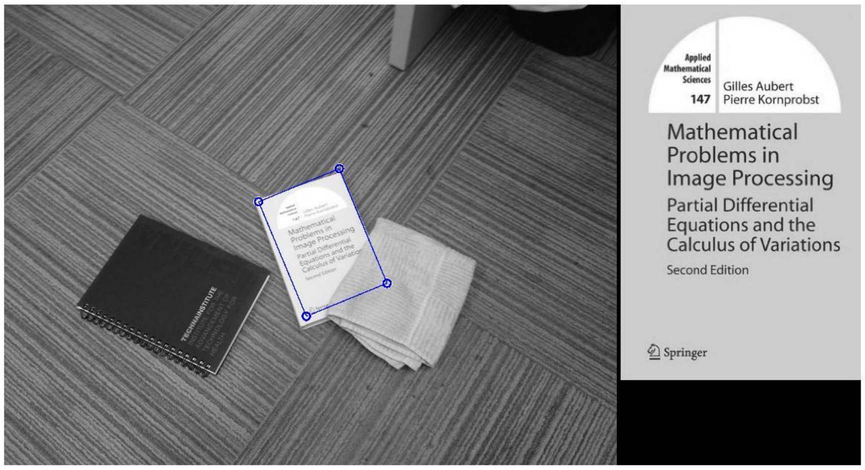
```
% 2 c, finding affine transform
% d2, img2,f2 -- findBook.jpg ; d1, img1,f1 -- book.jpg
match_matrix = zeros(size(d1,2),size(d2,2));
match_matrix(:,:) = pdist2(d1',d2');
max_two = zeros(size(match_matrix,1),2);
indeces_first_max = zeros(size(match_matrix,1),1);

[max_two(:,1),indeces_first_max(:)] = min(match_matrix,[],2);
for i = 1:length(indeces_first_max)
    match_matrix(i,indeces_first_max(i)) = 1000;
end
max_two(:,2) = min(match_matrix,[],2);

ratio = max_two(:,1)./max_two(:,2);

indices_picone = linspace(1,length(indeces_first_max),length(indeces_first_max))';
A = sortrows([ratio indeces_first_max indices_picone]);
ratio = A(:,1);
indeces_first_max = A(:,2);
indices_picone = A(:,3);

threshold = 0.8;
indeces_matched = (ratio < threshold) & indeces_first_max;

img_to_show = zeros(size(img2,1),size(img2,2)+size(img1,2)+5);
img_to_show(1:size(img2,1),1:size(img2,2)) = img2(:,:);
img_to_show(1:size(img1,1),size(img2,2)+6:size(img2,2)+5+size(img1,2)) = img1(:,:);
figure;
imshow(img_to_show);

K=30;

P_prime = zeros(2*K,1);
P = zeros(2*K,6);

for i = 1:K
    if(indeces_matched(i)~=0)
        P_prime(i*2-1) = f2(1,indeces_first_max(i));  % x'
        P_prime(i*2) = f2(2,indeces_first_max(i)); % y'

        P(i*2-1:i*2,:) = [f1(1,indices_picone(i)) f1(2,indices_picone(i)) 0 0 1 0;
                          0 0 f1(1,indices_picone(i)) f1(2,indices_picone(i)) 0 1];
    end
end

a = pinv(P'*P)*P'*P_prime;
```

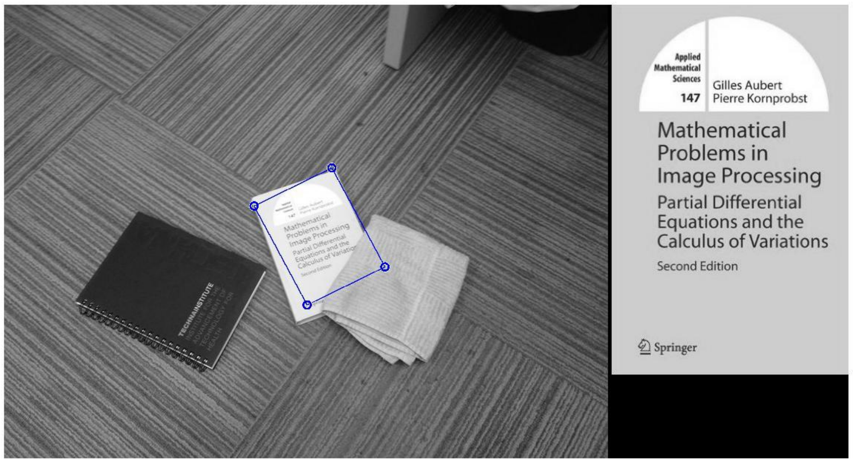
```
% 2 d, testing transform
test_points_img1 = [1 1; 1 320;499 320;499 1]; % y x
new_P = zeros(size(test_points_img1,1),6);
for i = 1:size(test_points_img1)
  new_P(i*2-1:i*2,:) = [test_points_img1(i,2) test_points_img1(i,1) 0 0 1 0;
                        0 0 test_points_img1(i,2) test_points_img1(i,1) 0 1];
end

matched_P = new_P*a;
for i=1:4
  viscircles([matched_P(2*i-1) matched_P(2*i)],5, 'color','b');
  if(i<4)
    lineX = [ matched_P(2*i-1),matched_P(2*(i+1)-1)];
```

2 d) The following images are produced with K = 10, 30, 70, and 100. The results between k=10 and 30 are similar, while as k becomes larger, the more inaccurate it becomes.

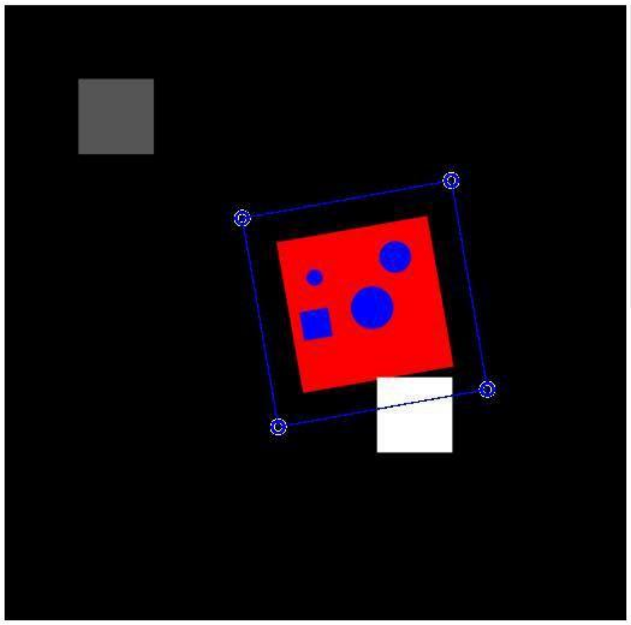
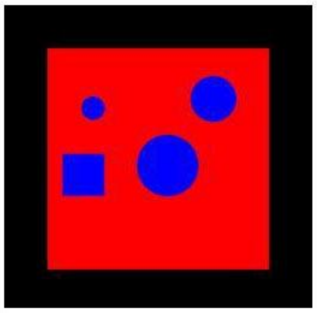





Applied Mathematical Sciences
147
Gilles Aubert
Pierre Kornprobst
Mathematical Problems in Image Processing
Partial Differential Equations and the Calculus of Variations
Second Edition
Springer


Applied Mathematical Sciences
147
Gilles Aubert
Pierre Kornprobst
Mathematical Problems in Image Processing
Partial Differential Equations and the Calculus of Variations
Second Edition
Springer

2 e) I fed the r, g, b channels of both images into the SIFT function and for both images got 3 sets of interest points positions and corresponding descriptors. Then I concatenated the descriptors, so now I have descriptors of the same length but more of them. My rationale is that by separately finding SIFT interest points and descriptors for each channel then combining the resulted descriptor, I can capture the change of each colour channel in terms location and scale, thus producing a similar effect of directly using gray-scaled images which is produced by simply taking the mean of the 3 channels. Here is my result and code to produce the combined descriptors for all channels. The matching part is the same from 2 d).

```
img1=double(imread('colourTemplate.png'))/255;
img2=double(imread('colourSearch.png'))/255;

addpath(genpath('./sift'));
[f1_r,d1_r]=sift(img1(:,:,1));
[f1_g,d1_g]=sift(img1(:,:,2));
[f1_b,d1_b]=sift(img1(:,:,3));

[f2_r,d2_r]=sift(img2(:,:,1));
[f2_g,d2_g]=sift(img2(:,:,2));
[f2_b,d2_b]=sift(img2(:,:,3));

f1 = [f1_r f1_g f1_b];
f2 = [f2_r f2_g f2_b];

d1 = [d1_r d1_g d1_b];
d2 = [d2_r d2_g d2_b];
```