ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: XỬ LÝ ẢNH

Đề tài: Tìm hiểu và cài đặt đặc trưng SURF

Giảng viên hướng dẫn: TS. Hoàng Văn Hiệp

Nhóm sinh viên:

Nguyễn Thị Oanh – 20163103

Nguyễn Hữu Tráng – 20164196

Hà Nội, tháng 7 năm 2020

MỤC LỤC

1. GIỚI THIỆU ĐẶC TRƯNG SURF

- Trong computer vision, speeded up robust features (SURF) là một thuật toán giúp phát hiện và mô tả đặc trưng cục bộ được sử dụng trong object recognition, image registration, classification, or 3D reconstruction,…Được lấy cảm hứng từ SIFT(scale-invariant feature transform), SURF nhanh hơn nhiều và được tác giả của nó tuyên bố rằng mạnh mẽ hơn trước các biến đổi hình ảnh khác nhau so với SIFT.
- Để phát hiện các điểm hấp dẫn, SURF sử dụng xấp xỉ định thức ma trận Hessian, có thể tính toán với ba phép toán số nguyên khi sử dụng hình ảnh tích hợp(integral image).
- Mô tả đặc trưng trong SURF dựa trên tổng phản hồi Haar wavelet xung quanh điểm hấp dẫn. Việc tính toán cũng rất nhanh khi sử dụng hình ảnh tích hợp.
- Các mô tả SURF đã được sử dụng để định vị và nhận dạng các đối tượng, người hoặc khuôn mặt, để tái tạo cảnh 3D, để theo dõi các đối tượng và trích xuất các điểm ưa thích.
- SURF lần đầu tiên được xuất bản bởi Herbert Bay, Tinne Tuytelaars và Luc Van Gool, và được trình bày tại Hội nghị châu Âu năm 2006 về Tầm nhìn máy tính. Một ứng dụng của thuật toán được cấp bằng sáng chế tại Hoa Kỳ. Phiên bản SURF "upright" (được gọi là U-SURF) không bất biến đối với xoay hình ảnh và do đó nhanh hơn để tính toán và phù hợp hơn cho ứng dụng trong đó máy ảnh vẫn nằm ngang hoặc ít hơn.
- Hình ảnh được chuyển thành tọa độ, sử dụng kỹ thuật kim tự tháp đa độ phân giải, để sao chép hình ảnh gốc với hình dạng Kim tự tháp Pyramidal Gaussian hoặc Laplacian để thu được hình ảnh có cùng kích thước nhưng băng thông giảm. Điều này đạt được hiệu ứng làm mờ đặc biệt trên ảnh gốc, được gọi là Scale-Space và đảm bảo rằng các điểm quan tâm là bất biến tỷ lệ.

2. CÀI ĐẶT SURF TRÊN MATLAB

2.1. Chuyển đổi ảnh đầu vào thành ảnh tích hợp(integral image)

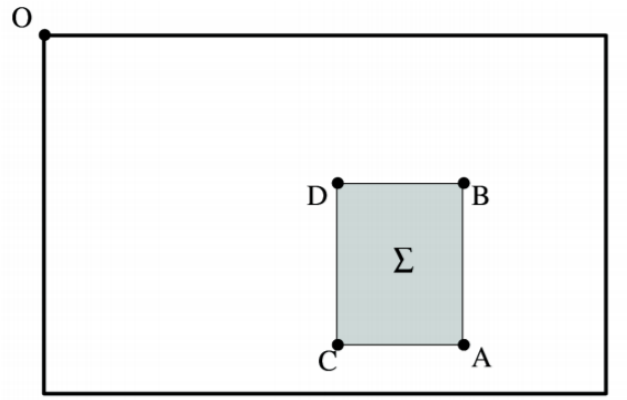
- Ảnh tích hợp(integral image) hay còn gọi là Summed-area table được giới thiệu vào năm 1984. Nó được sử dụng để tính toán một cách nhanh và có hiệu quả cho bài toán tính tổng các giá trị điểm ảnh trong toàn ảnh hay một vùng ảnh có hình chữ nhật. Trong SURF sẽ sử dụng tính chất này của ảnh tích hợp để tính nhanh phép nhân chập khi áp dụng box filters
- Công thức tính toán ảnh tích hợp: Giá trị tại mỗi điểm ảnh sẽ bằng tổng giá trị các điểm ảnh phía trên và bên trái của ảnh ban đầu.

$$I_{\Sigma}(\mathbf{x}) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j)$$

- Code Matlab chuyển đổi ảnh sang ảnh tích hợp:

```
function iimg = Convert_Integral_Image(img)
% Chuyen dinh dang anh dau vao thanh anh tich hop (integral
image)
% Input: img anh mau hoac anh xam
% Output: iimg anh tich hop(integral image) thu duoc tu img
    % Convert Image to double
    switch(class(img))
        case 'uint8'
            I=double(img)/255;
        case 'uint16'
            I=double(img)/65535;
        case 'int8'
            I=(double(img)+128)/255;
        case 'int16'
            I=(double(img)+32768)/65535;
        otherwise
            I=double(img);
    end

    % Convert Image to greyscale
    if(size(I,3)==3)
        cR = .2989; cG = .5870; cB = .1140;
        I=I(:,:,1)*cR+I(:,:,2)*cG+I(:,:,3)*cB;
    end

    % Make the integral image
    iimg = cumsum(cumsum(I,1),2);
end
```

- Tính toán tổng giá trị các điểm ảnh trong một vùng ảnh hình chữ nhật với integral image:



- Chú ý: ảnh đầu vào có kích thước MxN thì ảnh tích hợp sẽ có kích thước (M+1)x(N+1) (Do thêm 1 hàng và 1 cột toàn giá trị 0 và bên trên và bên trái ảnh). Lúc này, 1 vùng ảnh chữ nhật cần tính ở ảnh gốc khi ánh xạ sang ảnh ảnh tích hợp sẽ cần mở rộng về phía trên 1 hàng và phía bên trái 1 cột.

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

Original image

0	0	0	0	0	0
0	5	7	10	14	15
0	6	13	20	26	30
0	8	17	25	34	42
0	11	25	39	52	65
0	15	30	47	62	81

Integral image

- Code Matlab tính tổng 1 vùng ảnh hình chữ nhật trên ảnh tích hợp.

```
function rectSum = Calculate_Sum_Area(row,col, rows, cols,
iimg)
% Tinh toan tong gia tri cua 1 vung anh hinh chu nhat xac
dinh
% boi 4 tham so dau tien tren anh iimg
% Input:
%   row,col : 2 tham so xac dinh goc trai ben tren cua vung
can tinh
%    rows, cols: so hang va so cot cua vung can tinh, tinh
tu goc trai tren
% Output:
%   rectSum: Tong gia tri cac diem anh trong vung hinh chu
nhat can tinh
    % Get integer coordinates
    row=fix(row);
    col=fix(col);
    rows=fix(rows);
    cols=fix(cols);

    % Get the corner coordinates of the box integral
    r1 = min(row, size(iimg,1));
    c1 = min(col, size(iimg,2));
    r2 = min(row + rows, size(iimg,1));
    c2 = min(col + cols, size(iimg,2));

    % Get the values at the cornes of the box integral (fast
1D index look up)
    sx=size(iimg,1);
    A = iimg(max(r1+(c1-1)*sx,1));
    B = iimg(max(r1+(c2-1)*sx,1));
    C = iimg(max(r2+(c1-1)*sx,1));
    D = iimg(max(r2+(c2-1)*sx,1));

    % If coordinates are outside at the top or left, the
value must be zero
    A((r1<1)|(c1<1))=0;
    B((r1<1)|(c2<1))=0;
    C((r2<1)|(c1<1))=0;
    D((r2<1)|(c2<1))=0;

    % Minimum value of the integral is zero
    rectSum=max(0, A - B - C + D);
end
```

2.2. Ước lượng định thức của ma trận Hessian với box filters

- SURF sử dụng ma trận Hessian vì hiệu suất tính toán và độ chính xác cao.
- Thay vì sử dụng hai cách khác nhau để xác định vị trí và tỉ lệ (VD: Hessian-Laplace detector), SURF sử dụng định thức của ma trận Hessian cho cả hai.
- Với mỗi điểm **x** = *(x, y)* trong ảnh I, ma trận Hessian $H(\mathbf{x}, \sigma)$ tại x với scale σ được xác định như sau:

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

- $L_{xx}(\mathbf{x}, \sigma)$ = **convolution** $\left(\frac{\partial^2}{\partial x^2} g(\sigma), I(\mathbf{x})\right)$

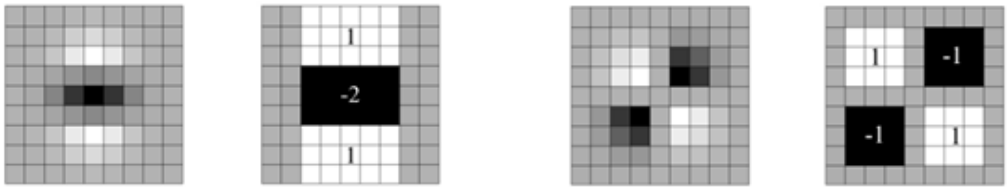
  $\frac{\partial^2}{\partial x^2} g(\sigma)$ Đạo hàm bậc 2 của hàm gaussian

  $I(\mathbf{x})$ Ảnh I tại vị trí **x**

- **Tương tự với** $L_{yy}(\mathbf{x}, \sigma)$ **và** $L_{xy}(\mathbf{x}, \sigma)$

- Để tính định thức ma trận Hessian, đầu tiên ta phải nhân chập ảnh với gausian sau đó lại tính đạo hàm bậc 2 → Chi phí tính toán lớn
- SIFT đã tìm được cách tính xấp sỉ LoG bằng DoG, SURF tính xấp xỉ định thức của ma trận Hessian thông qua box filters.

Box Filters 9x9 ~ sigma = 1.2



L_{yy} D_{yy} L_{xy} D_{xy}

- D_{xx} , D_{yy} , D_{xy} sẽ được tính bằng cách lấy giá trị các điểm ảnh nhân với giá trị tương ứng trong các điểm trên box filters như hình rồi lấy tổng tất cả trên box filters.(vùng xám thì giá trị điểm ảnh nhân với 0).
- Đinh thức Hessian tính bởi công thức sau:

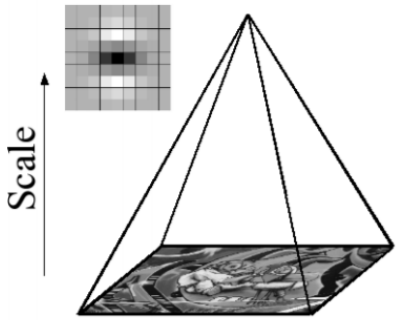
$$\det(\mathcal{H}_{\text{approx}}) = D_{xx}D_{yy} - (wD_{xy})^2.$$

$$w = \frac{|L_{xy}(1.2)|_F |D_{yy}(9)|_F}{|L_{yy}(1.2)|_F |D_{xy}(9)|_F} = 0.912... \simeq 0.9,$$

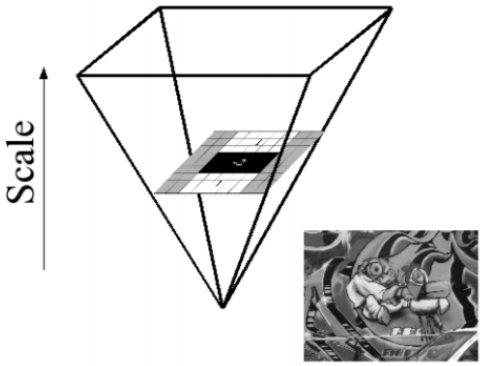
```
function                              layerData                              =
Calculate_Deteminant_Hessian(layerData,img)
% tinh det(H) va luu gia tri vao cau truc layerData
%  inputs,
%    layerData : layer can tinh gia det(H) va cac gia tri khac
%    img : anh dau vao de tinh cac gia tri
%  outputs,
%    layerData : layer da dien du du lieu
    step = fix( layerData.step); % step size for this filter
    b = fix((layerData.filter - 1) / 2 + 1);   % border for this
filter
    l = fix(layerData.filter / 3);    % lobe for this filter
(filter size / 3)(l0)
    w = fix(layerData.filter);        % filter size

    inverse_area = 1 / double(w * w);% normalisation factor

    [ac,ar]=ndgrid(0:layerData.width-1,0:layerData.height-1);
    ar=ar(:); ac=ac(:);

    % get the image coordinates
    r = int32(ar * step);
    c = int32(ac * step);

    % Compute response components
    Dxx =    Calculate_Sum_Area(r - l + 1, c - b, 2 * l - 1,
w,img) - Calculate_Sum_Area(r - l + 1, c - fix(l / 2), 2 * l -
1, l, img) * 3;
    Dyy =    Calculate_Sum_Area(r - b, c - l + 1, w, 2 * l -
1,img) - Calculate_Sum_Area(r - fix(l / 2), c - l + 1, l, 2 * l
- 1,img) * 3;
    Dxy = + Calculate_Sum_Area(r - l, c + 1, l, l,img) +
Calculate_Sum_Area(r + 1, c - l, l, l,img) ...
          - Calculate_Sum_Area(r - l, c - l, l, l,img) -
Calculate_Sum_Area(r + 1, c + 1, l, l,img);

    % Normalise the filter responses with respect to their size
    Dxx = Dxx*inverse_area;
    Dyy = Dyy*inverse_area;
    Dxy = Dxy*inverse_area;
    % Get the determinant of hessian response & laplacian sign
    layerData.responses = (Dxx .* Dyy - 0.9 * Dxy .* Dxy);
    layerData.laplacian = (Dxx + Dyy) >= 0;
end
```

2.3. Xây dựng Scale-space

- Với đặc trưng SIFT, Scale space là một kim tự tháp ảnh với kích thước ảnh giảm dần về phía đỉnh. Thường thường sẽ là giảm kích thước ảnh của ảnh phía dưới nó, tức là dùng đệ quy một phép xử lý giảm kích thước ảnh
- Với đặc trưng SURF, không sử dụng cách đệ quy giảm kích thước ảnh như SIFT để xây dựng scale space mà chọn cách tăng dần kích thước box filters, tăng dần bước lấy mẫu trên ảnh gốc với các octave. Như vậy sẽ tạo khả năng xử lý song song, tốc độ tính toán nhanh và chi phí là constant.

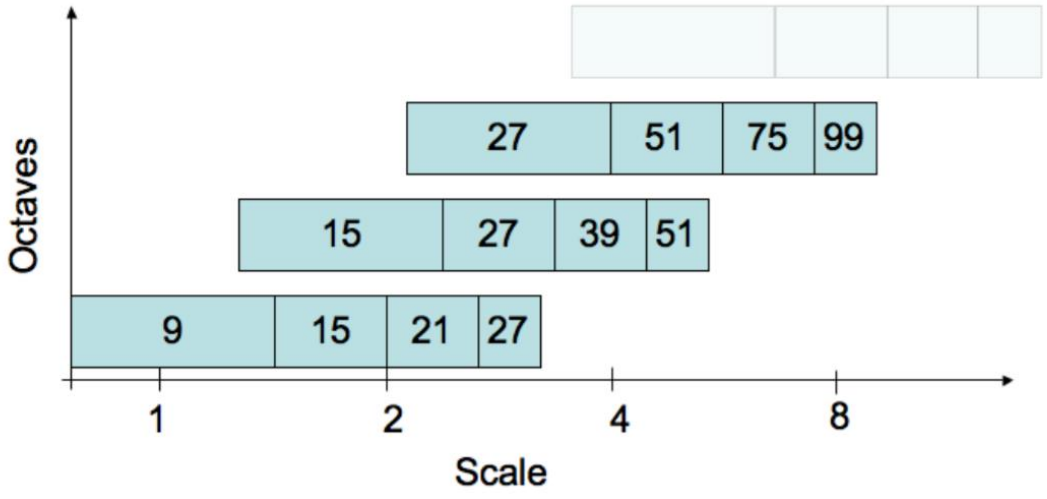


SIFT: Giảm kích thước ảnh bằng cách làm mịn nhiều lần bằng gaussian rồi lấy mẫu một cách đệ quy



SURF: Tăng dần kích thước box filter và đồng thời tăng bước lấy mẫu trên các octave một cách song song, chi phí tính toán là constant

- Với mỗi octave, thường sẽ dùng 4 bộ lọc box filters với kích thước tăng dần, minh họa như hình sau:(Mỗi octave sẽ có 4 layer ứng với 4 size box filters)

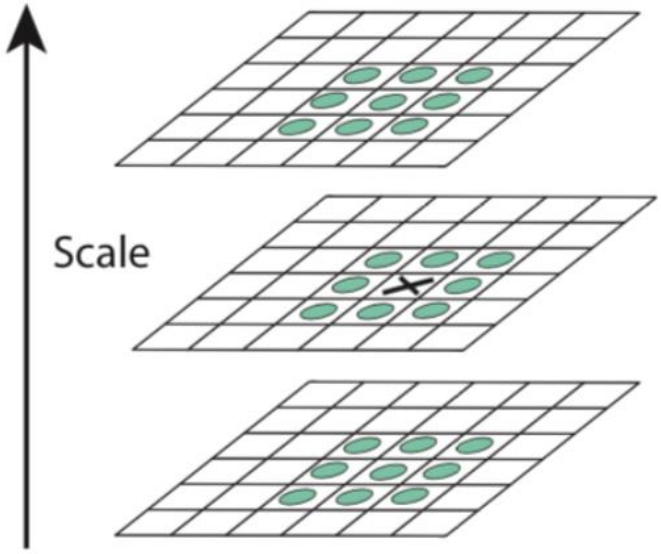


```
function layerData= Layer_Data(width, height, step, filter)
% xây dung cau truc du lieu luu tru thong tin cua 1 layer trong
scale space
% Input:
%   width, height: kich thuoc layer
%   step: Buoc lay mau(sampling step) cua layer nay
%   filter: kich thuoc box filter
% Output:
%   (structure)layerData{
%         width,
%         height,
%         step,
%         filter,
%        responses, : ma tran cot, kich thuoc width*height, luu
tru det(H)
%         laplacian  : ma tran cot, kich thuoc width*height
%     }
    width = floor(width);
    height = floor(height);
    step = floor(step);
    filter = floor(filter);

    layerData.width = width;
    layerData.height = height;
    layerData.step = step;
    layerData.filter = filter;
    layerData.responses = zeros(width * height,1);%luu det(H)
    layerData.laplacian = zeros(width * height,1);
end
```

```
function                              scaleSpaceMap                                =
Build_Scale_Space(iimg,n_octaves,init_sample)
% Calculate responses for the first n_octaves:
% Input:
%   iimg: anh dau vao danh dang integral image
%   n_octaves: so luong octave can tao
%   init_sample: buoc lay mau (sampling step)
% Output: scaleSpaceMap: thap scale space co data
    % cap phat bo nho luu tru scale space
    scaleSpaceMap = [];
    j = 0;
    % Lay kich thuoc anh
    w = (size(iimg,2)/init_sample);
    h = (size(iimg,1)/init_sample);
    step_sampling = init_sample;
    % Tao khung thap scale space
    if (n_octaves >= 1)
        j=j+1; scaleSpaceMap{j}=Layer_Data(w, h,step_sampling,9);
        j=j+1; scaleSpaceMap{j}=Layer_Data(w, h, step_sampling, 15);
        j=j+1; scaleSpaceMap{j}=Layer_Data(w, h, step_sampling, 21);
        j=j+1; scaleSpaceMap{j}=Layer_Data(w, h, step_sampling, 27);
    end
    if (n_octaves >= 2)
      j=j+1;scaleSpaceMap{j}=Layer_Data(w/2,h/2,step_sampling*2,39);
      j=j+1;scaleSpaceMap{j}=Layer_Data(w/2,h/2,step_sampling*2,51);
    end

    if (n_octaves >= 3)
     j=j+1;scaleSpaceMap{j}=Layer_Data(w/4,h/4,step_sampling*4,75);
     j=j+1;scaleSpaceMap{j}=Layer_Data(w/4,h/4,step_sampling*4,99);
    end

    if (n_octaves >= 4)
     j=j+1;scaleSpaceMap{j}=Layer_Data(w/8,h/8,step_sampling*8,147);
     j=j+1;scaleSpaceMap{j}=Layer_Data(w/8,h/8,step_sampling*8,195);
    end

    if (n_octaves >= 5)
        j=j+1;   scaleSpaceMap{j}=Layer_Data(w   /   16,   h   /   16,
step_sampling * 16, 291);
        j=j+1;   scaleSpaceMap{j}=Layer_Data(w   /   16,   h   /   16,
step_sampling * 16, 387);
    end
    % Extract responses from the image
    for i=1:length(scaleSpaceMap)
scaleSpaceMap{i}=Calculate_Deteminant_Hessian(scaleSpaceMap{i},iimg)
;
    end
end
```

2.4. Xác định các điểm hấp dẫn (interesting point)

- Trong SURF, điểm cực đại địa phương trong 3x3x3 điểm lân cận được coi là điểm hấp dẫn.



- Code Matlab tìm cực đại đại phương 3x3x3

```
function layerResponse = Get_Layer_Response(layer,row,
col,nomalize)
% tra ve 1 mang cac ptu co chi so trong mang index cua
det(H)
    scale=fix(layer.width/nomalize);
    % Clamp to boundary
    index=fix(scale*row) * layer.width + fix(scale*col)+1;
    index(index<1)=1;
index(index>length(layer.responses))=length(layer.response
s);
    layerResponse = layer.responses(index);
end
```

```
function                    isLocalMaximum                    =
Find_Local_Maximum(top,mid,bot,row,col,threshold)
% Tim cuc dai dia Phuong 3x3x3 tren 3 layer top, mid, bot
% Out: 1 ma tran cot kich thuoc bang kich thuoc det(H) cua
mid, luu tru gia tri bool de xac dinh co phai cuc dai dia
phuong hay khong
% bounds check
    layerBorder = fix((top.filter + 1) / (2 * top.step));
    bound_check_fail=(row   <=   layerBorder   |   row   >=
top.height - layerBorder | col <= layerBorder | col >=
top.width - layerBorder);
    normal = top.width;
    % check the candidate point in the middle layer is
above thresh
    candidate  =  Get_Layer_Response(mid,row,col,normal);%
lay tat ca det(H) cua mid
    treshold_fail = candidate < threshold;
    isLocalMaximum =(~bound_check_fail)&(~treshold_fail);
    for rr = -1:1
        for  cc = -1:1
             %  if any response in 3x3x3 is greater then
the candidate is not a maximum
             check1=Get_Layer_Response(top,row + rr, col
+ cc, normal) >= candidate;
             check2=Get_Layer_Response(mid,row + rr, col
+ cc, normal) >= candidate;
             check3=Get_Layer_Response(bot,row + rr, col
+ cc, normal) >= candidate;
             check4=(rr ~= 0 || cc ~= 0);
             check_all = ~(check1 | (check4 & check2) |
check3);
             isLocalMaximum=isLocalMaximum&check_all;
        end
    end
end
```

2.5. Xác định chính xác các điểm hấp dẫn bằng nội suy lân cận

- Sử dụng khai triển Taylor bậc 2 để nội suy lân cận.
- Ta sẽ tính cả Jacobian và Hessian để tính offset cho mỗi điểm tìm được ở bước 2.4. Nếu offset lớn hơn 0.5 thì nó thực sự gần với một điểm hấp dẫn ở cấp pixel khác, cho nên sẽ bị loại bỏ đi.

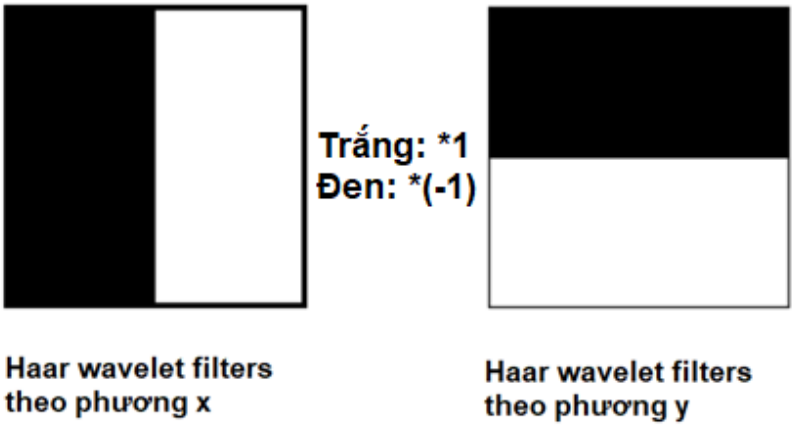
```
function [ipts, np] = Neighbor_Interpolation(r, c, t, m,
b, ipts,np)
% input:
%   r = row
%   c = col
%   t = top layer
%   m = mid layer
%   b = bot layer

    D = FastHessian_BuildDerivative(r, c, t, m, b);
    H = FastHessian_BuildHessian(r, c, t, m, b);

    %get the offsets from the interpolation: tinh phan bu
    Of = - H\D;
    O=[ Of(1, 1), Of(2, 1), Of(3, 1) ];

    %get the step distance between filters
    filterStep = fix((m.filter - b.filter));

    %If point is sufficiently close to the actual extremum:
phan bu phai
    %nho hon 0.5
    if (abs(O(1)) < 0.5 && abs(O(2)) < 0.5 && abs(O(3)) <
0.5)
        np=np+1;
        ipts(np).x = double(((c + O(1))) * t.step);
        ipts(np).y = double(((r + O(2))) * t.step);
        ipts(np).scale = double(((2/15) * (m.filter + O(3)
* filterStep)));
        ipts(np).laplacian = fix(Get_Laplacian(m,r,c,t));
    end

end
```

```
function D=FastHessian_BuildDerivative(r,c,t,m,b)
    dx=(Get_Response(m,r,c+1,t)-Get_Response(m,r,c-1,t))/2;
    dy=(Get_Response(m,r+1,c,t)-Get_Response(m,r-1,c,t))/2;
    ds =(Get_Response(t,r,c)-Get_Response(b,r,c,t))/2;
    D = [dx;dy;ds];
end

function H=FastHessian_BuildHessian(r, c, t, m, b)
    v = Get_Response(m, r, c, t);
    dxx = Get_Response(m,r,c +1,t)+Get_Response(m,r,c-1,t)-2*v;
    dyy = Get_Response(m,r+1,c,t)+Get_Response(m,r-1,c,t)-2*v;
    dss = Get_Response(t,r,c) + Get_Response(b,r,c,t)-2*v;
    dxy = (Get_Response(m,r + 1, c + 1, t) - Get_Response(m,r + 1, c
- 1, t) - Get_Response(m,r - 1, c + 1, t) + Get_Response(m,r - 1, c -
1, t)) / 4;
    dxs = (Get_Response(t,r, c + 1) - Get_Response(t,r, c - 1) -
Get_Response(b,r, c + 1, t) + Get_Response(b,r, c - 1, t)) / 4;
    dys = (Get_Response(t,r + 1, c) - Get_Response(t,r - 1, c) -
Get_Response(b,r + 1, c, t) + Get_Response(b,r - 1, c, t)) / 4;

    H = zeros(3,3);
    H(1, 1) = dxx;
    H(1, 2) = dxy;
    H(1, 3) = dxs;
    H(2, 1) = dxy;
    H(2, 2) = dyy;
    H(2, 3) = dys;
    H(3, 1) = dxs;
    H(3, 2) = dys;
    H(3, 3) = dss;
end

function response = Get_Response(a,row, column,b)
    if(nargin<4)
        scale=1;
    else
        scale=fix(a.width/b.width);
    end
    response=a.responses(fix(scale*row)       *       a.width       +
fix(scale*column)+1);
end
function laplacian = Get_Laplacian(a,row, column,b)
    if(nargin<4)
        scale=1;
    else
        scale=fix(a.width/b.width);
    end
    laplacian=a.laplacian(fix(scale*row)       *       a.width       +
fix(scale*column)+1);
end
```

```
function ipts = Get_Interest_Point(iimg)
% Tim cac diem hap dan tren anh iimg
    % filter index map
    filter_map = [0,1,2,3;
                  1,3,4,5;
                  3,5,6,7;
                  5,7,8,9;
                  7,9,10,11]+1;

    % Set tham so mac dinh
    n_octave = 5; % so octave
    init_sample = 2; % buoc lay mau mac dinh
    threshold = 0.001;
    np=0; ipts=struct;
    % Build the scale space map
    scaleSpaceMap                                                    =
Build_Scale_Space(iimg,n_octave,init_sample);
    % Tim cuc dai trong dia phuong scale and space
    for o = 1:n_octave
        for i = 1:2
            bot = scaleSpaceMap{filter_map(o,i)};
            mid = scaleSpaceMap{filter_map(o,i+1)};
            top = scaleSpaceMap{filter_map(o,i+2)};

            % loop over middle response layer at density of the
most
            % sparse layer (always top), to find maxima across
scale and space
            [col,row]=ndgrid(0:top.width-1,0:top.height-1);
            row=row(:); col=col(:);

            % p = array(index cac diem local maximum)
            p                                                        =
find(Find_Local_Maximum(top,mid,bot,row,col,threshold));

            % Xac dinh chinh xac diem hap dan bang noi suy lan
can
            for j=1:length(p)
                index=p(j);
                [ipts,np]=Neighbor_Interpolation(row(index),
col(index), top, mid, bot, ipts,np);
            end
        end
    end
```

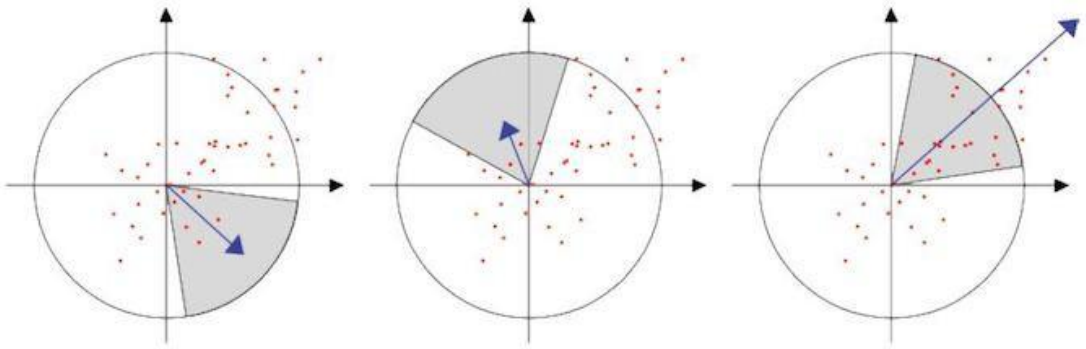
2.6. Tính hướng tham chiếu cho các điểm hấp dẫn

- Để có thể thực hiện được bất biến với phép xoay, SURF tính hướng(orientation) cho các điểm hấp dẫn đã tìm được ở trên.
- Đầu tiên sẽ tính Haar-wavelet responses theo phương x và y trong một vòng tròn bán kính 6s quanh điểm hấp dẫn đang xét (s là scale ứng với điểm hấp dẫn đang xét và được chọn là step sampling). Ở đây ta lại dùng ảnh tích hợp để lọc nhanh.




```
function resX=IntegralImage_HaarX(row, column, size, img)
% Tinh Haar-wavelet responses theo phuong X
    resX= Calculate_Sum_Area(row-size/2,column, size, size
/ 2, img)...
        - Calculate_Sum_Area(row - size / 2, column - size
/ 2, size, size / 2, img);
end


function resY=IntegralImage_HaarY(row, column, size, img)
% Tinh Haar-wavelet responses theo phuong Y
    resY= Calculate_Sum_Area(row, column - size / 2, size
/ 2, size , img)...
        - Calculate_Sum_Area(row - size / 2, column - size
/ 2, size / 2, size , img);
end
```

- Sau đó tính tổng các phản hồi Haar-wavelet theo 2 phương x và y trong một khu vực quét, sau đó lại thay đổi hướng quét (60^o) và quét tiếp cho đến khi tìm được giá trị lớn nhất, đó chính là hướng cần tìm(orientation)

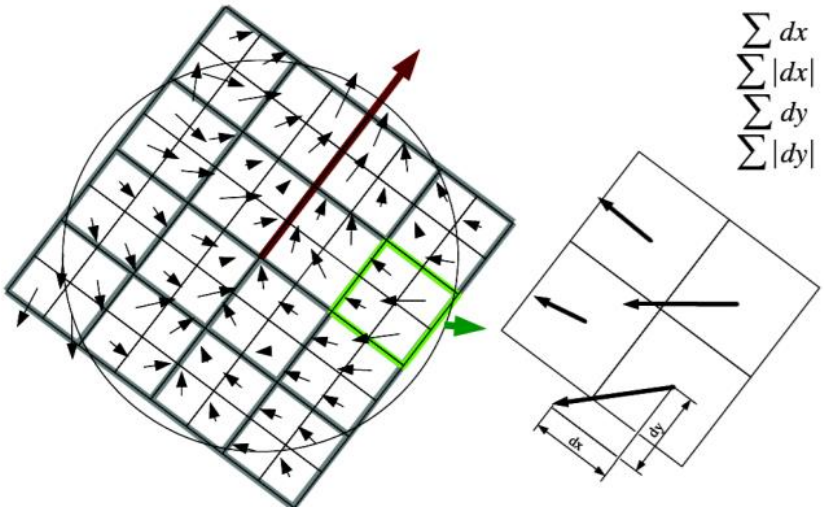


```
function orientation = Get_Orientation(ip,img)
% Tinh huong cua diem hap dan
% Input:
%   ip: diem hap dan(interesting point) = Ipts[i]
%   img: anh dau vao de tinh Haar-wavelet responses

    % Ma tran trong so: nhung diem gan ip dang xet hon se anh huong
nhieu
    % hon den huong
    gauss25  =  [0.02350693969273  0.01849121369071  0.01239503121241
0.00708015417522 0.00344628101733 0.00142945847484 0.00050524879060;
                0.02169964028389  0.01706954162243  0.01144205592615
0.00653580605408 0.00318131834134 0.00131955648461 0.00046640341759;
                0.01706954162243  0.01342737701584  0.00900063997939
0.00514124713667 0.00250251364222 0.00103799989504 0.00036688592278;
                0.01144205592615  0.00900063997939  0.00603330940534
0.00344628101733 0.00167748505986 0.00069579213743 0.00024593098864;
                0.00653580605408  0.00514124713667  0.00344628101733
0.00196854695367 0.00095819467066 0.00039744277546 0.00014047800980;
                0.00318131834134  0.00250251364222  0.00167748505986
0.00095819467066 0.00046640341759 0.00019345616757 0.00006837798818;
                0.00131955648461  0.00103799989504  0.00069579213743
0.00039744277546 0.00019345616757 0.00008024231247 0.00002836202103];
    gauss25=gauss25(:);


    % Get rounded InterestPoint data
    X = round(ip.x);
    Y = round(ip.y);
    S = round(ip.scale);

    % calculate haar responses for points within radius of 6*scale
```

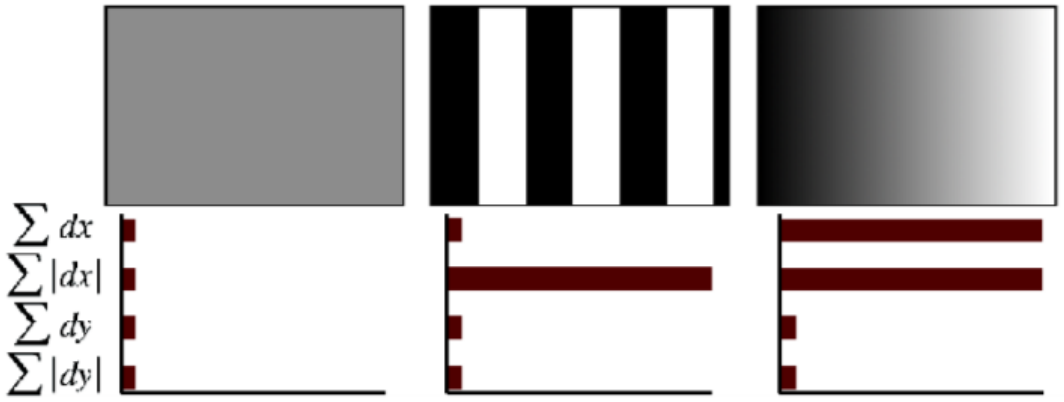
```
% calculate haar responses for points within radius of 6*scale
    [j,i]=ndgrid(-6:6,-6:6);
    j=j(:); i=i(:); check=(i.^2 + j.^2 < 36); j=j(check); i=i(check);

    % Get gaussian filter (by mirroring gauss25)
    id = [ 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6 ];
    gauss = gauss25(id(i + 6 + 1) + id(j + 6 + 1) *7+1);

    resX = gauss .* IntegralImage_HaarX(Y + j * S, X + i * S, 4 * S,
img);
    resY = gauss .* IntegralImage_HaarY(Y + j * S, X + i * S, 4 * S,
img);
    Ang =  mod(atan2(resY, resX),2*pi);

    % loop slides pi/3 window around feature point
    ang1 = 0:0.15:(2 * pi);
    ang2 = mod(ang1+pi/3,2*pi);

    % Repmat is used to check for all angles (x direction) and
    % all responses (y direction) without for-loops.
    cx=length(Ang); cy=length(ang1);
    ang1=repmat(ang1,[cx 1]);
    ang2=repmat(ang2,[cx 1]);
    Ang =repmat(Ang,[1 cy]);
    resX =repmat(resX,[1 cy]);
    resY =repmat(resY,[1 cy]);

    % determine whether the point is within the window
    check1= (ang1 < ang2) & (ang1 < Ang) & (Ang < ang2);
    check2= (ang2 < ang1) & ( ((Ang > 0) & (Ang < ang2)) | ((Ang >
ang1) & (Ang < pi)) );
    check=check1|check2;

    sumX =  sum(resX.*check,1);
    sumY =  sum(resY.*check,1);

    % Find the most dominant direction
    R=sumX.^2+ sumY.^2;
    [buf,ind]=max(R);
    orientation =  mod(atan2(sumY(ind), sumX(ind)),2*pi);

    if(0)
        pica=zeros(13,13);
        pica(i+7+(j+6)*13)=Ang(:,1);
        imshow(pica,[0 2*pi]);
    end

end
```

2.7. Xây dựng bộ mô tả đặc trưng cho các điểm hấp dẫn(descriptor)

- Đầu tiên xây dựng một khu vực hình vuông xung quanh điểm hấp dẫn đang xét và định hướng theo hướng tìm được ở bước 2.6 với kích thước 20s.
- Sau đó chia khu vực hình vuông này thành các vùng con hình vuông nhỏ hơn có kích thước 4x4. Với mỗi vùng con, ta tính phản ứng Haar wavelet tại 5x5 điểm mẫu cách đều nhau. Để đơn giản, ta gọi dx, dy là các phản hồi Haar wavelet theo các phương x và y. Để tăng cường độ cho các biến dạng hình học và lỗi cục bộ, dx và dy được đặt trọng số với Gaussian(sigma = 3.3s) tại trung tâm của điểm hấp dẫn đang xét.



- Sau đó tính tổng các phản hồi dx và dy trong vùng con, ta được 2 trường đầu tiên trong vector mô tả. Để thể hiện thông tin về sự thay đổi cường độ, trích suất thêm giá trị tuyệt đối của dx và dy và đưa vào vector mô tả. Như vậy, với mỗi vùng con 4x4 có 1 vector mô tả 4 chiều dạng **V = (∑ dx, ∑ dy, ∑|dx|, ∑|dy|).** Do đó, khi gộp tất cả các tiểu vùng lại ta sẽ có 1 vector mô tả 64 chiều.



```
function descriptor = Get_Descriptor(ip, img)
% Tinh gia descriptor: 64 chieu
% Input:
%   ip = ipts[i]
%   img: anh dau vao
% Output
%   descriptor: vector 64 chieu, bieu dien diem hap dan

    X = round(ip.x);
    Y = round(ip.y);
    S = round(ip.scale);

    co = cos(ip.orientation);
    si = sin(ip.orientation);

    % Basis coordinates of samples, if coordinate 0,0, and scale 1
    [lb,kb]=ndgrid(-4:4,-4:4); lb=lb(:); kb=kb(:);

    %Calculate descriptor for this interest point
    [jl,il]=ndgrid(0:3,0:3); il=il(:)'; jl=jl(:)';

    ix = (il*5-8);
    jx = (jl*5-8);

    % 2D matrices instead of double for-loops, il, jl
    cx=length(lb); cy=length(ix);
    lb=repmat(lb,[1 cy]); lb=lb(:);
    kb=repmat(kb,[1 cy]); kb=kb(:);
    ix=repmat(ix,[cx 1]); ix=ix(:);
    jx=repmat(jx,[cx 1]); jx=jx(:);

    % Coordinates of samples (not rotated)
    l=lb+jx; k=kb+ix;

    %Get coords of sample point on the rotated axis
    sample_x = round(X + (-l * S * si + k * S * co));
    sample_y = round(Y + (l * S * co + k * S * si));

    %Get the gaussian weighted x and y responses
    xs = round(X + (-(jx+1) * S * si + (ix+1) * S * co));
    ys = round(Y + ((jx+1) * S * co + (ix+1) * S * si));

    gauss_s1 = SurfDescriptor_Gaussian(xs - sample_x, ys - sample_y,
2.5 * S);
    rx = IntegralImage_HaarX(sample_y, sample_x, 2 * S,img);
    ry = IntegralImage_HaarY(sample_y, sample_x, 2 * S,img);

    %Get the gaussian weighted x and y responses on the aligned axis
```

```
%Get the gaussian weighted x and y responses on the aligned axis
    rrx = gauss_s1 .* (-rx * si + ry * co);  rrx=reshape(rrx,cx,cy);
    rry = gauss_s1 .* ( rx * co + ry * si);  rry=reshape(rry,cx,cy);

    % Get the gaussian scaling
    cx = -0.5 + il + 1; cy = -0.5 + jl + 1;
    gauss_s2 = SurfDescriptor_Gaussian(cx - 2, cy - 2, 1.5);

    dx = sum(rrx,1);
    dy = sum(rry,1);
    mdx = sum(abs(rrx),1);
    mdy = sum(abs(rry),1);
    dx_yn = 0; mdx_yn = 0;
    dy_xn = 0; mdy_xn = 0;

    descriptor=[dx;dy;mdx;mdy].* repmat(gauss_s2,[4 1]);

    len = sum((dx.^2 + dy.^2 + mdx.^2 + mdy.^2 + dx_yn + dy_xn +
mdx_yn + mdy_xn) .* gauss_s2.^2);

    %Convert to Unit Vector
    descriptor= descriptor(:) / sqrt(len);
function surfGaussian= SurfDescriptor_Gaussian(x, y, sig)
    surfGaussian = 1 / (2 * pi * sig^2) .* exp(-(x.^2 + y.^2) / (2 *
sig^2));
end
```

```
function ipts = Descriptor_Interesting_Points(ipts, img)
% Them orientation va descriptor vao cau truc ipts
% Input:
%   ipts: danh sach cac diem hap dan tim duoc
%   img: anh dau vao(integral image)
% Output:
%  ipts[]: danh sach cac diem hap dan da co du thong tin orientation,
descriptor
    if (isempty(fields(ipts))), return; end
    for i=1:length(ipts)
       ip=ipts(i);
       % descriptor size
       ip.descriptorLength = 64;
       % tinh orientation
       ip.orientation=Get_Orientation(ip,img);
       % dien thong tin cho SURF descriptor
       ip.descriptor=Get_Descriptor(ip, img);
       ipts(i).orientation=ip.orientation;
       ipts(i).descriptor=ip.descriptor;
    end
end
```

2.8. So khớp các điểm hấp dẫn(matching)

- Tính khoảng cách Euclid giữa các điểm hấp dẫn tìm được ở 2 ảnh với nhau.
- Hiển thị 10 điểm khớp nhau nhất.

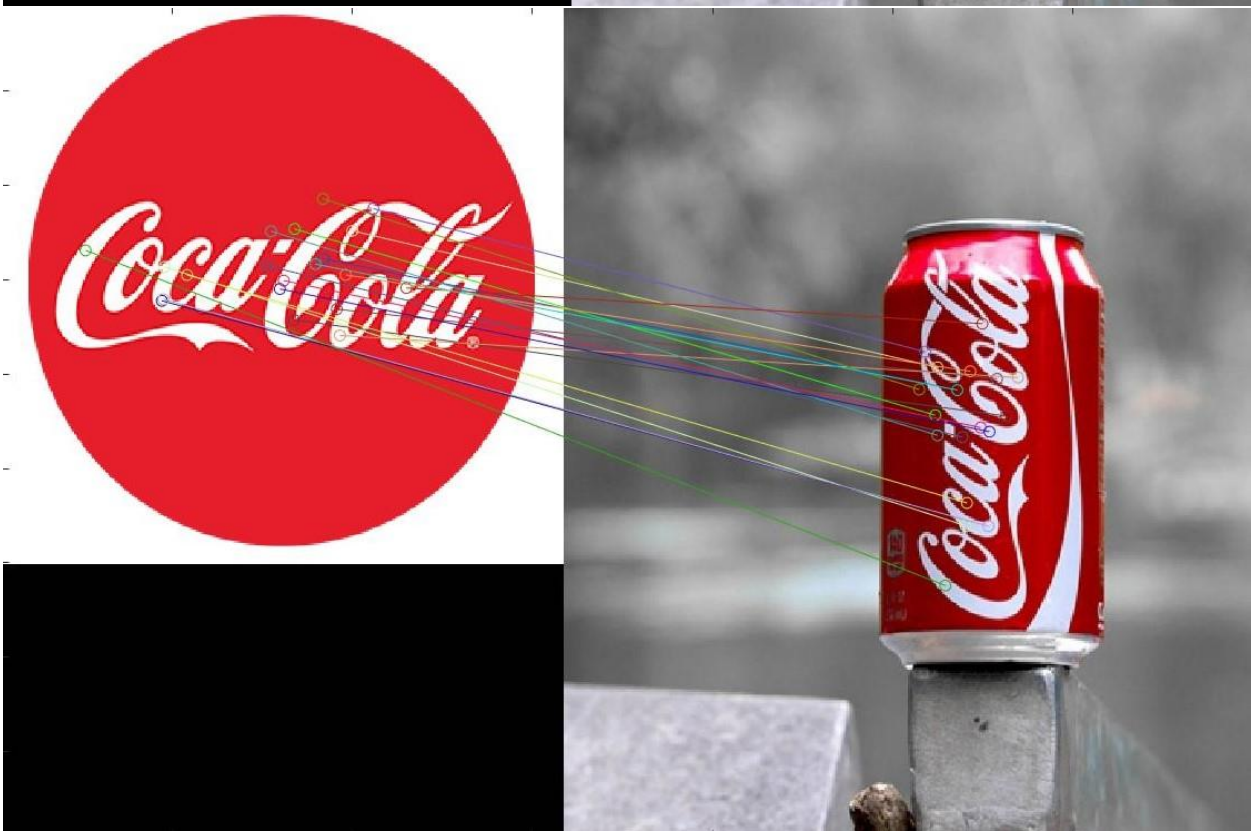
```
% Doc anh dau vao
I1 = imread('coca_logo.png');
I2 = imread('coca_cola4.jpg');
figure,imshow(I1);
figure,imshow(I2);

% convert sang anh tich hop
iI1 = Convert_Integral_Image(I1);
iI2 = Convert_Integral_Image(I2);

% Xac dinh cac diem hap dan
ipts1 = Get_Interest_Point(iI1);
ipts2 = Get_Interest_Point(iI2);

if(~isempty(ipts1))
    ipts1 = Descriptor_Interesting_Points(ipts1,iI1);
end

if(~isempty(ipts2))
    ipts2 = Descriptor_Interesting_Points(ipts2,iI2);
end

% Put the landmark descriptors in a matrix
  D1 = reshape([ipts1.descriptor],64,[]);
  D2 = reshape([ipts2.descriptor],64,[]);

% Find the best matches
  err=zeros(1,length(ipts1));
  cor1=1:length(ipts1);
  cor2=zeros(1,length(ipts1));
  for i=1:length(ipts1),
      distance=sum((D2-repmat(D1(:,i),[1
length(ipts2)])).^2,1);
      [err(i),cor2(i)]=min(distance);
  end
% Sort matches on vector distance
  [err, ind]=sort(err);
  cor1=cor1(ind);
  cor2=cor2(ind);


% Show both images
```

```
% Show both images
  figure;
  height_1 = size(I1,1);
  width_1 = size(I1,2);
  height_2 = size(I2,1);
  width_2 = size(I2,2);

  total_width = width_1 + width_2;
  if(height_1>height_2)
      total_height = height_1;
  else
      total_height = height_2;
  end

  combined_image = ones(total_height, total_width, 3);
  combined_image(1:height_1,1:width_1,:) = I1;
  combined_image(1:height_2,(width_1+1):(width_2+width_1),:)
= I2;
  imagesc(double(combined_image)/double(255));
  colormap(gray(256));
  hold on;

% Show the best matches
  for i=1:20
      c=rand(1,3);
      plot([ipts1(cor1(i)).x
ipts2(cor2(i)).x+size(I1,2)],[ipts1(cor1(i)).y
ipts2(cor2(i)).y],'-','Color',c)
      plot([ipts1(cor1(i)).x
ipts2(cor2(i)).x+size(I1,2)],[ipts1(cor1(i)).y
ipts2(cor2(i)).y],'o','Color',c)
  end
```

3. KẾT QUẢ THỬ NGHIỆM

- **Các hình thử nghiệm theo cặp: hình trên vẽ 10 điểm, hình dưới vẽ 20 điểm best match**


Coca-Cola
Coca-Cola


Coca-Cola
Coca-Cola


ORIGINAL TASTE
YI NGUYEN BAN
Coca-Cola
ORIGINAL TASTE
YI NGUYEN BAN
Coca-Cola

- **Hình trên: Hessian threshold = 0.0002; Hình dưới : Hessian threshold = 0.001**

KẾT LUẬN

Sau một thời gian tìm hiểu và cài đặt thì kết quả nhóm đã cài đặt thành công thuật toán SURF. Kết quả khá khả quan trong đa số trường hợp. Tuy nhiên do thời gian có hạn nên còn nhiều thiếu sót, đặc biệt ở phần matching cũng như chưa xây dựng được ứng dụng hoàn chỉnh và thực tế nào. Nhóm em rất mong nhận được những ý kiến của thầy và các bạn đọc bản báo cáo này. Em xin thay mặt nhóm chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

1. Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "Speeded Up Robust Features", ETH Zurich, Katholieke Universiteit Leuven.

2. https://github.com/herbertbay/SURF

3. OpenSURF https://www.mathworks.com/matlabcentral/fileexchange/28300-opensurf-including-image-warp