VIỆN CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Đề tài:

TÌM HIỂU ĐẶC TRƯNG SIFT

Giảng viên hướng dẫn: TS. Hoàng Văn Hiệp

Sinh viên thực hiện: Lê Thanh Tùng - MSSV: 20145095

Hà Nội 05/2018

LỜI GIỚI THIỆU

Con người thu nhận thông tin từ môi trường ngoài thông qua các giác quan, trong đó thị giác đóng vai trò quan trọng nhất trong quá trình nhận thức. Sự phát triển của phần cứng máy tính về phương diện thu nhận, lưu trữ, xử lý, hiển thị đã vạch ra nhiều định hướng mới cho sự phát triển phần mềm nói chung và lĩnh vực xử lý ảnh nói riêng. Cùng với kỹ thuật đồ họa, xử lý ảnh đóng vai trò quan trọng trong các hệ thống tương tác người máy.

Nhận dạng đối tượng là một bài toán điển hình của lĩnh vực xử lý ảnh, với mục tiêu giúp cho máy tính có thể nhận thức được môi trường ngoài giống như con người thông qua "thị giác". Bài toán này có thể được áp dụng trong việc phát hiện, nhận dạng, theo dõi hay tìm kiếm tự động các đối tượng trong thực tế, điển hình như việc giám sát an ninh cho các khu vực quan trọng: ngân hàng, khu vực chính trị, quân sự... Một trong các phương pháp để nhận dạng đối tượng đó là trích chọn ra các đặc trưng từ ảnh, và so khớp các đặc trưng tìm được đó với các đặc trưng có trong tập mẫu để có thể rút ra kết luận. Khi nhắc đến thuật toán trích chọn điểm đặc trưng cục bộ có tính chất bất biến với một số phép biến đổi ảnh, chắc hẳn SIFT (Scale Invariant Feature Transform) là thuật toán được biết đến rộng rãi bởi tính chính xác và ứng dụng cao trong nhiều bài toán. Bài báo phân tích chi tiết thuật toán được viết trên IPOL (Image Processing Online) journal, việc cài đặt và thử nghiệm thuật toán trong bài tập lớn cũng được thực hiện theo bài báo đó.

Trong quá trình tìm hiểu và thực hiện, em có thể sẽ gặp phải nhiều thiếu sót, em mong nhận được phản hồi và góp ý từ thầy để bài làm được trở nên đầy đủ và hoàn thiện hơn.

MỤC LỤC

DANH MỤC HÌNH VẼ

1. GIỚI THIỆU ĐẶC TRƯNG SIFT

SIFT (Scale Invariant Feature Transform) là phát minh của tác giả David Lowe được đưa ra vào năm 1999, có nhiều ưu điểm và hiệu quả cao trong các bài toán đối sánh và nhận dạng đối tượng trong ảnh. SIFT tìm ra các điểm đặc trưng cục bộ (local features hay local interesting points), có tính chất bất biến đối với phép dịch (translation), phép xoay (rotation) và phép zoom-out (scaling) làm cơ sở cho đối sánh, nhận dạng đối tượng trong ảnh. SIFT cũng được chứng minh là ít bị ảnh hưởng bởi các phép biến đổi ảnh, như là thay đổi nhỏ về điểm quan sát, nhiễu, mờ, độ tương phản, biến dạng của cảnh vật và có đủ tính phân biệt dùng cho các ứng dụng so khớp (matching). Thuật toán SIFT bao gồm 2 thao tác liên tiếp và độc lập: phát hiện các điểm đặc trưng (detect interesting points – keypoints) và xây dựng bộ mô tả (descriptor) cho các điểm tương ứng đó.

Về nguyên tắc hoạt động, SIFT tìm ra các điểm keypoint từ biểu diễn ảnh dưới nhiều kích thước khác nhau, với mức độ bị làm mờ (blurred) tăng dần. Đối với mỗi keypoint, thuật toán SIFT sẽ tính toán tìm ra hướng tham chiếu (reference orientation) thông qua miền lân cận bao quanh keypoint đó. Tiếp đến, thuật toán sẽ xây dựng bộ mô tả dựa trên miền bao quanh keypoint với hướng tham chiếu được tính ra, do đó đặc trưng SIFT có tính chất bất biến đối với phép xoay ảnh (rotation). Bộ mô tả là 1 vector có 128 chiều, được dùng cho mục đích so khớp các ảnh đối tượng với nhau.

SIFT bao gồm khá nhiều bước. Để có tính bất biến đối với kích thước (scale), ta cần xây dựng không gian ảnh với các kích thước khác nhau, áp dụng mặt nạ Gaussian với tham số khác nhau lên các ảnh đó, gọi là **Gaussian scale-space**. Nhân chập Gaussian đóng vai trò mô phỏng việc ảnh được chụp ở các khoảng cách khác nhau (càng xa ảnh càng mờ hơn), phần này sẽ được trình bày cụ thể trong phần 2a.

Việc phát hiện và xác định vị trí các keypoint bao gồm việc tính toán Difference of Gaussians (DoG) từ scale-space ban đầu (phần 2b), tìm kiếm các điểm cực trị từ không gian DoG (phần 2c), sau đó là điều chỉnh để có vị trí với

độ chính xác cao hơn dựa trên nội suy lân cận (phần 2d). Sau đó, các điểm nhiễu và các điểm nằm trên các cạnh sẽ bị loại bỏ dựa trên việc lấy ngưỡng (phần 2e và 2f).

SIFT có tính bất biến với phép xoay nhờ việc tính ra hướng tham chiếu cho mỗi keypoint từ hướng gradient lân cận xung quanh keypoint (phần 2g). Cuối cùng, phân bố của gradient quanh miền theo hướng tham chiếu của keypoint được dùng để tạo ra descriptor cho keypoint đó (phần 2h). Bên cạnh đó, một thuật toán so khớp (matching) cũng được đưa ra làm ví dụ cho ứng dụng của SIFT (phần 2i).

Hình 1: Ví dụ 2 kết quả thu được với đặc trưng SIFT (implement trên MATLAB)

2. CÀI ĐẶT THUẬT TOÁN SIFT TRÊN MATLAB

a. Tính toán Gaussian scale-space

Bước 1: Dùng nội suy song tuyến tính (bilinear interpolation) tạo ảnh khởi tạo có kích thước gấp đôi ảnh đầu vào

Bước 2: Lặp với số lượng octave muốn tạo:

+ Áp mặt nạ Gauss lên ảnh với các hệ số khác nhau, bao nhiêu hệ số tùy vào số lượng scale mong muốn cho mỗi octave.

+ Giảm kích thước ảnh đi 2 lần và và tiếp tục lặp lại.

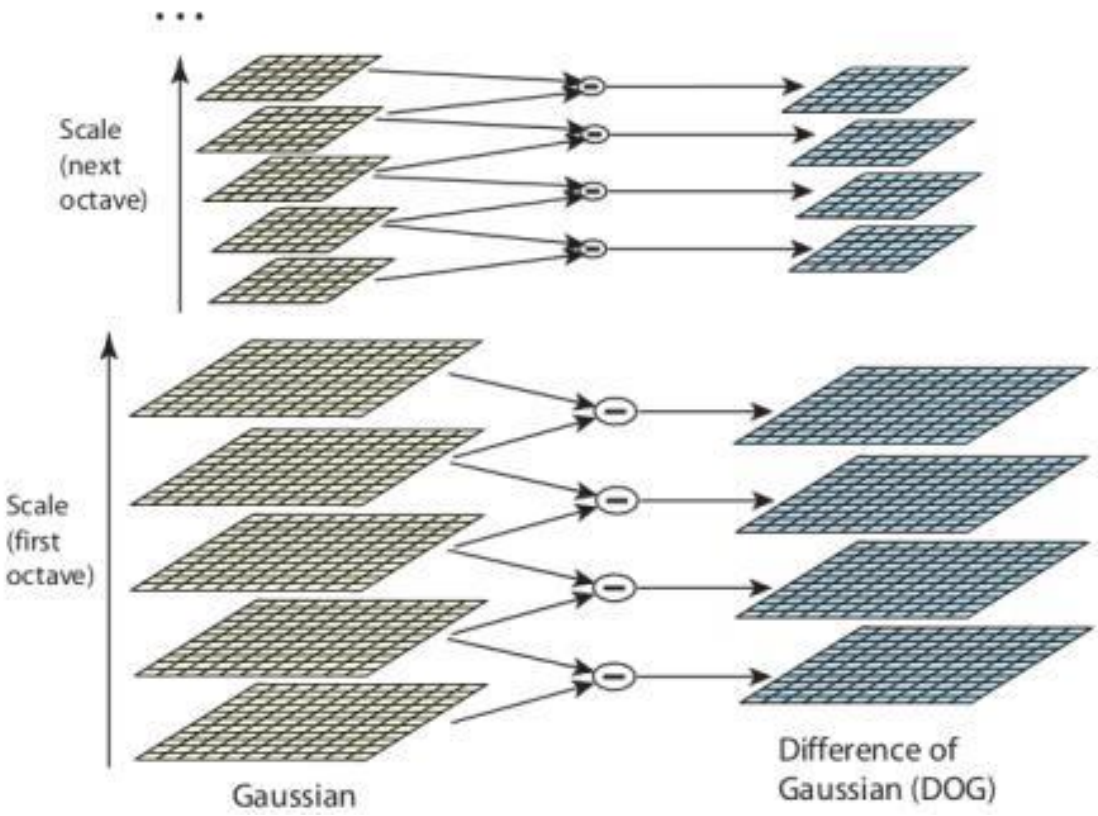
- Mã nguồn trên MATLAB:

```
%Ham tinh toan nhan chap Gauss voi ma tran I
function r = gaussian_filter(I, sigma, kernel_size)
    if ~exist('kernel_size', 'var')
        kernel_size = round(4*sigma)*2+1;
    end

    if(kernel_size < 1)
        kernel_size = 1;
    end
    h = fspecial('gaussian', kernel_size, sigma);
    r = filter2(h, I, 'same');
end


function all_octaves = scale_space(I)
    sigma_min = 0.8;  %tham so sigma khoi tao
    delta_min = 0.5;  %ban dau anh zoom len 2 lan
    n_spo = 3;        %so luong scale moi octave
    n_oct = 5;        %so luong octave
    sigma_in = 0.5;   %tham so Gauss nhan chap voi anh dau
vao

    total_scales = n_spo + 3; %moi octave co 3 scale bo sung
    if(size(I, 3) > 1)
        I = rgb2gray(I);
    End
    %tao anh dau vao kich thuoc gap doi
    I_init = imresize(I, 1/delta_min, 'bilinear');
    H_init = fspecial('gaussian', 3, 0.5);
    %nhan chap Gauss voi anh dau vao
```

```
    I_init = filter2(H_init, I_init, 'same');

    curr_sigma = sigma_min;
    init_image = I_init;
    prev_image = I_init;
    octaves = cell(n_oct, 1);
    %Tinh toan tren tung octave
    for o=1:n_oct
        octaves{o} = zeros(size(init_image, 1),
size(init_image, 2), total_scales);
        for s=1:total_scales
            blured_image = gaussian_filter(prev_image,
curr_sigma);
            prev_image = blured_image;
            disp(['Octave number: ' num2str(o) ' with blur
level ' num2str(s) ' and sigma value = '
num2str(curr_sigma)]);
            octaves{o}(:, :, s) = blured_image;
            curr_sigma  = sigma_min * (2^(s/n_spo+o-1));
        end
        %tinh lai sigma cho anh dau tien cua octave
        curr_sigma  = sigma_min * (2^o);
        %giam kich thuoc anh di 2 lan
        init_image =
reduce_a_half(octaves{o}(:,:,total_scales-3));
        prev_image = init_image;
    end

    all_octaves = octaves;

    function r = reduce_a_half(I)
        r=I(1:2:end,1:2:end);
    end
end
```

b. Tính toán Difference of Gaussians (DoG)

- Đối với mỗi octave, lấy 2 ma trận liên tiếp trừ đi nhau:



Hình 2: Gaussian Scale Space và DoG Scale Space

- Mã nguồn trên MATLAB:

```
function DOG_octaves = calculate_DOG(octaves)
    DOG = cell(size(octaves, 1), 1);
    for i=1:size(octaves, 1)
        DOG{i} = zeros(size(octaves{i}, 1), size(octaves{i},
2), size(octaves{i}, 3)-1);
        for j=2:size(octaves{i},3)
            DOG{i}(:,:,j-1) = octaves{i}(:,:,j) -
octaves{i}(:,:,j-1);
        end
    end
    DOG_octaves = DOG;
end
```

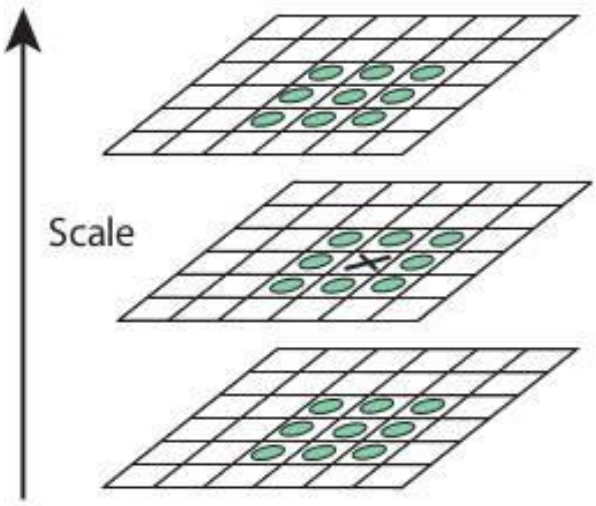
c. Tìm các điểm đặc trưng (keypoints)

- Duyệt qua từng DoG octave:

+ Tìm ra các điểm cực trị (cực đại hoặc cực tiểu) trong vùng 26 giá trị lân cận

+ Không xét scale đầu tiên và cuối cùng của mỗi DoG octave, các điểm nằm trên biên ma trận

+ Nếu giá trị tuyệt đối của cực trị tìm được < 0.8 * C_DoG thì loại bỏ (C_DoG mặc định là 0.015)



Hình 3: Tìm kiếm các điểm cực trị trên DoG Scale Space

- Mã nguồn trên MATLAB:

```
function keypoints = calculate_key_points(DOG)
    C_DOG = 0.015;
    C_EDGE = 10;
    delta_min = 0.5;
    sigma_min = 0.8;
    n_spo = 3;

    count = 0;
    keypoint_abs = {};
    for o=1:size(DOG, 1)
        for l=2:(size(DOG{o}, 3)-1)
            for r = 2:size(DOG{o},1)-1
                for c = 2:size(DOG{o},2)-1
                    is_keypoint = false;
                        if DOG{o}(r,c,l) > DOG{o}(r-1,c,l)
           && DOG{o}(r,c,l) > DOG{o}(r-1,c-1,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c-1,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c-1,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c+1,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c+1,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c+1,l) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c-1,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c-1,l-1) ...
```

```
        && DOG{o}(r,c,l) > DOG{o}(r+1,c-1,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c+1,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c+1,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c+1,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c,l-1) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c-1,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c-1,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c-1,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r+1,c+1,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c+1,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r-1,c+1,l+1) ...
        && DOG{o}(r,c,l) > DOG{o}(r,c,l+1)

                        if(is_keypoint == false)
                            count = count + 1;
                            is_keypoint = true;
                        end
                    end

                    if DOG{o}(r,c,l) < DOG{o}(r-1,c,l)
           && DOG{o}(r,c,l) < DOG{o}(r-1,c-1,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c-1,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c-1,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c+1,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c+1,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c+1,l) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c-1,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c-1,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c-1,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c+1,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c+1,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c+1,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c,l-1) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c-1,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c-1,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c-1,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r+1,c+1,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c+1,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r-1,c+1,l+1) ...
        && DOG{o}(r,c,l) < DOG{o}(r,c,l+1)

                        if(is_keypoint == false)
                            count = count + 1;
                            is_keypoint = true;
                        end
```

```
                end

                if(is_keypoint==true)
                    if(abs(DOG{o}(r,c,l)) < 0.8*C_DOG)
                        count = count - 1;
                        is_keypoint = false;
                    end
                end
                ...
```

d. Xác định chính xác điểm đặc trưng dựa trên nội suy lân cận

- Khởi tạo α = [1 1 1]
- Lặp tối đa 5 lần hoặc đến khi max($|\alpha|$) < 0.6 thì dừng:

  + Tính $\bar{g}$ và $\bar{H}$ theo công thức:

$$\bar{\boldsymbol{g}} = \frac{1}{2}\begin{bmatrix} DoG^o_{s+1,m,n} - DoG^o_{s-1,m,n} \\ DoG^o_{s,m+1,n} - DoG^o_{s,m-1,n} \\ DoG^o_{s,m,n+1} - DoG^o_{s,m,n-1} \end{bmatrix}; \bar{\boldsymbol{H}} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{12} & h_{22} & h_{23} \\ h_{13} & h_{23} & h_{33} \end{bmatrix}$$

$$h_{11} = DoG^o_{s+1,m,n} + DoG^o_{s-1,m,n} - 2DoG^o_{s,m,n};$$
$$h_{12} = (DoG^o_{s+1,m+1,n} - DoG^o_{s+1,m-1,n} - DoG^o_{s-1,m+1,n} + DoG^o_{s-1,m-1,n})/4;$$
$$h_{22} = DoG^o_{s,m+1,n} + DoG^o_{s,m-1,n} - 2DoG^o_{s,m,n};$$
$$h_{13} = (DoG^o_{s+1,m,n+1} - DoG^o_{s+1,m,n-1} - DoG^o_{s-1,m,n+1} + DoG^o_{s-1,m,n-1})/4;$$
$$h_{33} = DoG^o_{s+1,m,n} + DoG^o_{s-1,m,n} - 2DoG^o_{s,m,n};$$
$$h_{23} = (DoG^o_{s,m+1,n+1} - DoG^o_{s,m+1,n-1} - DoG^o_{s,m-1,n+1} + DoG^o_{s,m-1,n-1})/4;$$

  + Tính α^* và ω:

$$\boldsymbol{\alpha}^* = -\bar{H}^{-1}\bar{g}; \boldsymbol{\omega} = DoG^o_{s,m,n} - \frac{1}{2}\bar{g}^T\bar{H}^{-1}\bar{g}$$

  + Tính lại vị trí chính xác của điểm đặc trưng:

$$\boldsymbol{\delta_{oe}} = \delta_{min} * 2^{o-1}$$

$$(\boldsymbol{\sigma}, \boldsymbol{x}, \boldsymbol{y}) = (2^{o-1} * \sigma_{min} * 2^{\frac{\alpha_1^* + s}{n_{spo}}}, \delta_{oe}(\alpha_2^* + m), \delta_{oe}(\alpha_3^* + n))$$

+ Cập nhật lại giá trị s, m, n:

$$(\boldsymbol{s}, \boldsymbol{m}, \boldsymbol{n}) = ([s + \alpha_1^*], [m + \alpha_2^*], [n + \alpha_3^*])$$

$[.]$: *ký hiệu hàm round*

- Nếu đã kết thúc 5 lần lặp, $\max(|\alpha_1^*|, |\alpha_2^*|, |\alpha_3^*|) \geq 0.6$ thì keypoint bị loại bỏ
- <u>Mã nguồn trên MATLAB:</u>

```
        ...
        l_n = l;
        r_n = r;
        c_n = c;
        x_n = 0;
        y_n = 0;
        sigma_n = 0;
        w = 0;
        H = zeros(3, 3);
        if(is_keypoint==true)
            g = zeros(3, 1);
            num_tries = 1;
            alpha = ones(1, 3);
            while(num_tries < 6 && max(abs(alpha)) >= 0.6)
                H(1, 1) = DOG{o}(r_n,c_n,l_n+1) +
DOG{o}(r_n,c_n,l_n-1) - 2*DOG{o}(r_n,c_n,l_n);
                H(2, 2) = DOG{o}(r_n+1,c_n,l_n) + DOG{o}(r_n-
1,c_n,l_n) - 2*DOG{o}(r_n,c_n,l_n);
                H(3, 3) = DOG{o}(r_n,c_n+1,l_n) +
DOG{o}(r_n,c_n-1,l_n) - 2*DOG{o}(r_n,c_n,l_n);
                H(1, 2) = (DOG{o}(r_n+1,c_n,l_n+1) -
DOG{o}(r_n-1,c_n,l_n+1) - DOG{o}(r_n+1,c_n,l_n-1) +
DOG{o}(r_n-1,c_n,l_n-1))/4;
                H(1, 3) = (DOG{o}(r_n,c_n+1,l_n+1) -
DOG{o}(r_n,c_n-1,l_n+1) - DOG{o}(r_n,c_n+1,l_n-1) +
DOG{o}(r_n,c_n-1,l_n-1))/4;
                H(2, 3) = (DOG{o}(r_n+1,c_n+1,l_n) -
DOG{o}(r_n+1,c_n-1,l_n) - DOG{o}(r_n-1,c_n+1,l_n) +
DOG{o}(r_n-1,c_n-1,l_n))/4;
                H(2, 1) = H(1, 2);
                H(3, 1) = H(1, 3);
                H(3, 2) = H(2, 3);
                g(1, 1) = (DOG{o}(r_n,c_n,l_n+1) -
DOG{o}(r_n,c_n,l_n-1))/2;
                g(2, 1) = (DOG{o}(r_n+1,c_n,l_n) - DOG{o}(r_n-
1,c_n,l_n))/2;
                g(3, 1) = (DOG{o}(r_n,c_n+1,l_n) -
DOG{o}(r_n,c_n-1,l_n))/2;
                alpha = -inv(H)*g;
                w = DOG{o}(r_n,c_n,l_n) - 0.5*g'*inv(H)*g;
```

```
                delta_oe = delta_min*2^(o-1);
                if(max(abs(alpha))<0.6)
                    sigma_n = 2^(o-
1)*sigma_min*2^((alpha(1)+l_n)/n_spo);
                    x_n = delta_oe*(alpha(2) + r_n);
                    y_n = delta_oe*(alpha(3) + c_n);
                end
                l_n = round(l_n + alpha(1));
                r_n = round(r_n + alpha(2));
                c_n = round(c_n + alpha(3));
                if(l_n <= 1 || r_n <= 1 || c_n <= 1 || l_n >=
(size(DOG{o}, 3)-1) || r_n >= size(DOG{o},1)-1 || c_n >=
size(DOG{o},2)-1)
                    is_keypoint = false;
                    count = count - 1;
                    break;
                end
                num_tries = num_tries + 1;
            end

            if(is_keypoint==true)
                if(max(abs(alpha)) >= 0.6)
                    is_keypoint = false;
                    count = count - 1;
                end
            end
        end
        ...
```

e. Loại bỏ các điểm đặc trưng do nhiễu

- Nếu $|\omega| \geq C_{DOG}$ thì keypoint được giữ lại, nếu không bị loại bỏ
- Mã nguồn trên MATLAB:

```
                    ...
                    %Loai bo cac keypoint do nhieu
                    if(is_keypoint==true)
                        if(abs(w) < C_DOG)
                            is_keypoint = false;
                            count = count - 1;
                        end
                    end
                    ...
```

f. Loại bỏ các điểm đặc trưng nằm trên các cạnh

- Tính tr(H) và det(H) với $H = \begin{bmatrix} h_{22} & h_{23} \\ h_{23} & h_{33} \end{bmatrix}$ đã có từ bước tính $\bar{H}$.
- Nếu $\frac{tr(H)^2}{\det(H)} < \frac{(C_{edge}+1)^2}{C_{edge}}$ (C_{edge} mặc định $= 10$) thì keypoint được giữ lại, nếu trái lại sẽ bị loại bỏ.
- Mã nguồn trên MATLAB:

```
                    ...
                    %Loai bo keypoint nam tren cac canh
                    if(is_keypoint==true)
                        tr = H(2, 2) + H(3, 3);
                        determinant_H = H(2, 2) * H(3, 3) -
H(2, 3)*H(2, 3);
                        ratio = tr^2/determinant_H;
                        threshold = ((C_EDGE+1)^2)/C_EDGE;
                        if(ratio >= threshold ||
determinant_H < 0)
                            is_keypoint = false;
                            count = count - 1;
                        end
                    end

                    if(is_keypoint==true)
                       keypoint_abs{count} = [o l_n sigma_n
x_n y_n];
                    end
                end
            end
        end
    end

    result = cell(2,1);
    result{1} = keypoint_abs;
    result{2} = count;
    keypoints = result;
end
```

g. Tính hướng tham chiếu cho mỗi điểm đặc trưng

- Tính toán góc và độ lớn của vector gradient trên các octave **Gaussian scale space**

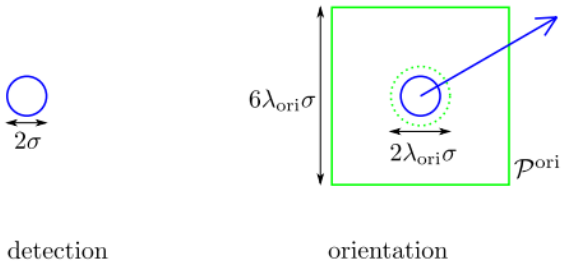
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

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

+ Có thể thực hiện bằng phép nhân chập với 2 kernel:

$$w_x(x,y) = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}; \qquad w_y(x,\ y) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- Tại mỗi keypoint, xét lân cận $6\lambda_{ori}\sigma$ với λ_{ori} mặc định là 1.5, σ là tham số mặt nạ Gauss được áp dụng.

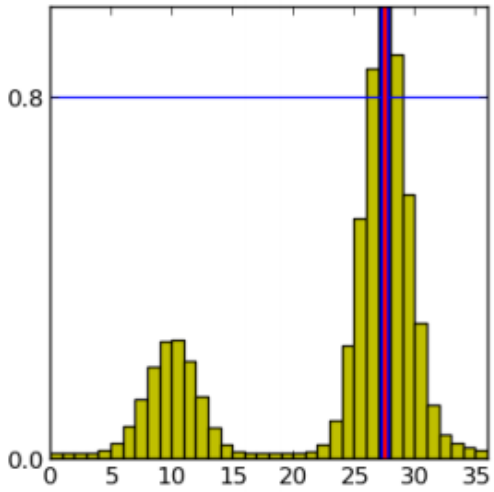


Hình 4: Tính hướng tham chiếu trong lân cận keypoint

- Tính toán histogram theo hướng trong lân cận điểm đặc trưng:

  + Khởi tạo histogram, mặc định 36 bins (góc từ 0->2π) đều bằng 0.

  + Với mỗi điểm trong lân cận, nhân độ lớn vector gradient với 1 trọng số Gauss, rồi cộng giá trị đó vào bin tương ứng với góc vector gradient của điểm đó. Làm tương tự với các điểm còn lại.



Hình 5: Histogram theo các hướng của vùng lân cận keypoint

- Nhân chập histogram với bộ lọc [1,1,1]/3
- Từ histogram, tính ra hướng tham chiếu tại bin histogram thỏa mãn: *h(k) > h(k-1)* và *h(k) > h(k+1)* và *h(k) >= t*max(h)*, trong đó t mặc định = 0.8

$$\theta_{key} = \frac{2\pi(k-1)}{n_{bins}} + \frac{\pi}{n_{bins}}\left(\frac{h_{k-1} - h_{k+1}}{h_{k-1} - 2h_k + h_{k+1}}\right)$$

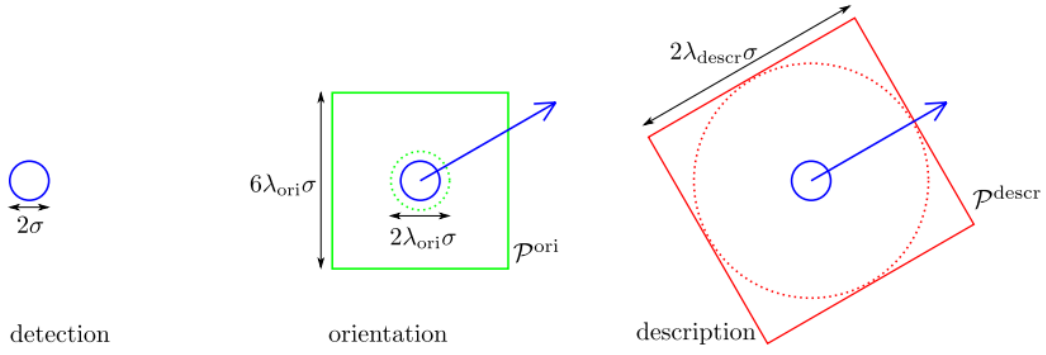
- Mã nguồn trên MATLAB:

```
function grad_and_theta_keys = define_orientation(key_points,
octaves, I)
    orient_magn = cell(size(octaves,1),size(octaves{1},4),2);
    [height_I, width_I] = size(I(:,:,1));
    lambda = 1.5;
    delta_min = 0.5;
    n_bins = 36;

    count = 0;
    o_count = 0;
    key_points_n = cell(key_points{2}, 1);
    for o = 1:size(octaves,1)
        for s = 1:size(octaves{o},3)
            filter_diff_x = [0 0 0; -1 0 1; 0 0 0];
            diff_x_mat = imfilter(octaves{o}(:,:,s),
filter_diff_x);

            filter_diff_y = [0 1 0; 0 0 0; 0 -1 0];
```

```
            diff_y_mat = imfilter(octaves{o}(:,:,s),
filter_diff_y);

            magn_mat = sqrt(diff_x_mat.*diff_x_mat +
diff_y_mat.*diff_y_mat);
            orient_mat = atan2(diff_y_mat, diff_x_mat);

            orient_magn{o}{s}{1} = magn_mat;
            orient_magn{o}{s}{2} = orient_mat;
        end
    end

    theta_keys = cell(key_points{2}, 1);
    for kp=1:key_points{2}
        theta_keys{kp} = 0;
        key_point = key_points{1}{kp};
        o_key = key_point(1);
        s_key = key_point(2);
        x_key = key_point(4);
        y_key = key_point(5);
        sigma_key = key_point(3);
        if(x_key >= 3*lambda*sigma_key && x_key <= height_I -
3*lambda*sigma_key && ...
                y_key >= 3*lambda*sigma_key && y_key <=
width_I - 3*lambda*sigma_key)
            hist = zeros(n_bins,1);
            m_from = round((x_key-
3*lambda*sigma_key)/(delta_min*2^(o_key-1)));
            m_to =
round((x_key+3*lambda*sigma_key)/(delta_min*2^(o_key-1)));
            n_from = round((y_key-
3*lambda*sigma_key)/(delta_min*2^(o_key-1)));
            n_to =
round((y_key+3*lambda*sigma_key)/(delta_min*2^(o_key-1)));
            m_from = max(m_from, 1);
            n_from = max(n_from, 1);
            for m=m_from:m_to
                for n=n_from:n_to
                    temp_1 = m*delta_min*2^(o_key-1) - x_key;
                    temp_2 = n*delta_min*2^(o_key-1) - y_key;
                    temp_3 = 2*((lambda*sigma_key)^2);
                    c_ori_mn = exp(-(temp_1^2 +
temp_2^2)/temp_3) * orient_magn{o_key}{s_key}{1}(m,n);
                    bin_ori_mn =
round(n_bins/(2*pi)*mod(orient_magn{o_key}{s_key}{2}(m,n),
2*pi)+0.5);
                    hist(bin_ori_mn) = hist(bin_ori_mn) +
c_ori_mn;
                end
            end

            smoothing_kernel = [1/3 1/3 1/3];
            hist = conv(hist, smoothing_kernel);
```

```
            t = 0.8;
            for k=2:n_bins-1
                if(hist(k)>hist(k-1) && hist(k)>hist(k+1) &&
hist(k)>=t*max(hist))
                    if(count == o_count)
                        count = count + 1;
                    end
                    theta_keys{count} = 2*pi*(k-1)/n_bins +
pi/n_bins*((hist(k-1)-hist(k+1))/(hist(k-1)-
2*hist(k)+hist(k+1)));
                    key_points_n{count} = key_point;
                end
            end
            o_count = count;
        end
    end
    result = cell(4, 1);
    result{1} = theta_keys;
    result{2} = orient_magn;
    result{3} = key_points_n;
    result{4} = count;
    grad_and_theta_keys = result;
end
```

h. Xây dựng bộ mô tả điểm đặc trưng (keypoint descriptor)



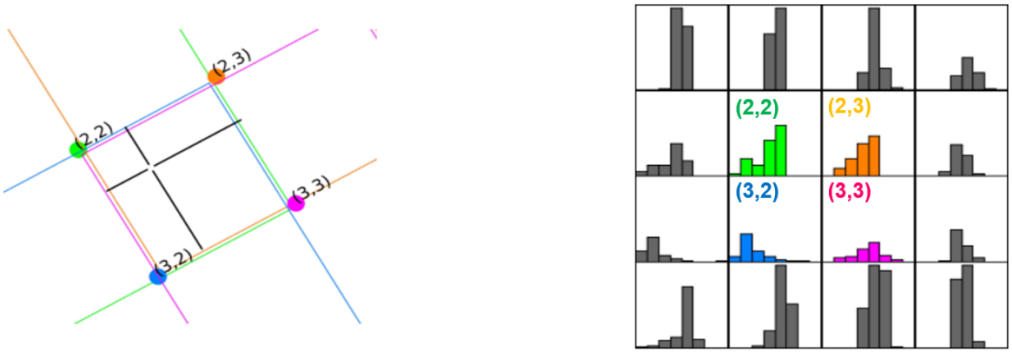
Hình 6: Bộ mô tả (descriptor) được xây dựng quanh lân cận theo hướng tham chiếu của keypoint

- Xét lân cận có kích thước $2\lambda_{descr}\sigma$ (mặc định $\lambda_{descr} = 6$) xung quanh điểm đặc trưng theo hướng tham chiếu đã tính từ bước trên:

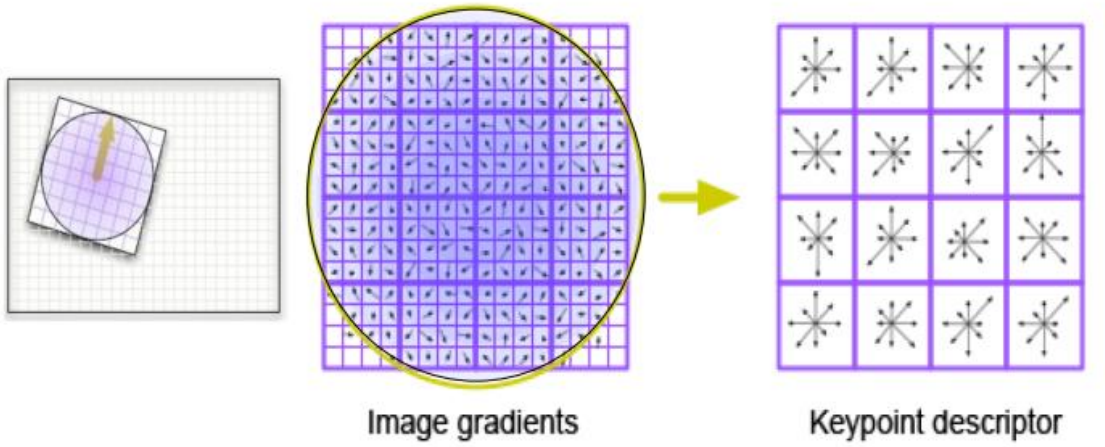
+ Chia lân cận đó thành vùng 4x4, mỗi vùng khởi tạo 1 histogram 8 bins ban đầu đều là 0

+ Tại mỗi điểm trong vùng lân cận, tính toán hướng gradient được chuẩn hóa theo hướng tham chiếu và độ lớn gradient có trọng số Gaussian

+ Cộng giá trị tìm được vào bin của histogram tương ứng của các vùng

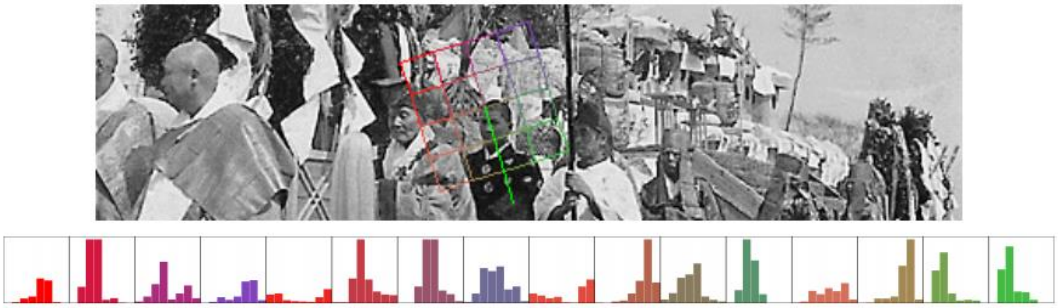


Hình 7: Histogram tại từng vùng trong lưới 4x4 lân cận keypoint



Hình 8: Minh họa xây dựng descriptor từ vector gradient

- Tính toán và chuẩn hóa vector đặc trưng thu được từ 4x4 histogram bên trên, thu được descriptor có 4x4x8 = 128 chiều



Hình 9: Tạo descriptor từ 4x4 histogram thu được vector 128 chiều

- Mã nguồn trên MATLAB:

```
function descriptors = sift_descriptor(num_keypoints,
key_points, orient_magn, theta_keys, I)
    [height_I, width_I] = size(I(:,:,1));

    lambda_descr = 6;
    n_hist = 4;
    n_ori = 8;
    delta_min = 0.5;

    f = cell(num_keypoints, 1);
    abs_keypoint = cell(num_keypoints, 1);
    %f = {};
    count = 0;
    for kp=1:num_keypoints
        key_point = key_points{kp};
        o_key = key_point(1);
        s_key = key_point(2);
        sigma_key = key_point(3);
        x_key = key_point(4);
        y_key = key_point(5);
        temp = sqrt(2)*lambda_descr*sigma_key;
        if(x_key >= temp && x_key <= height_I - temp...
                && y_key >= temp && y_key <= width_I - temp)
            hist = zeros(n_hist, n_hist, n_ori);
            temp = temp * (n_hist+1)/n_hist;
            m_from = round((x_key -
temp)/(delta_min*2^(o_key-1)));
            m_to = round((x_key + temp)/(delta_min*2^(o_key-
1)));
            n_from = round((y_key -
temp)/(delta_min*2^(o_key-1)));
            n_to = round((y_key + temp)/(delta_min*2^(o_key-
1)));
            m_to = min(size(orient_magn{o_key}{s_key}{2},1),
m_to);
            n_to = min(size(orient_magn{o_key}{s_key}{2},2),
n_to);
            m_from = max(m_from, 1);
            n_from = max(n_from, 1);
            for m=m_from:m_to
                for n=n_from:n_to
                    x_mn = ((m*delta_min*2^(o_key-1)-
x_key)*cos(theta_keys{kp}) + ...
                            (n*delta_min*2^(o_key-1)-
y_key)*sin(theta_keys{kp}))/sigma_key;
                    y_mn = (-(m*delta_min*2^(o_key-1)-
x_key)*sin(theta_keys{kp}) + ...
                            (n*delta_min*2^(o_key-1)-
y_key)*cos(theta_keys{kp}))/sigma_key;
```

```
                    if(max(abs(x_mn), abs(y_mn)) <
lambda_descr*(n_hist+1)/n_hist)
                        theta_mn =
orient_magn{o_key}{s_key}{2}(m,n) - mod(theta_keys{kp},
2*pi);
                        temp_1 = m*delta_min*2^(o_key-1) -
x_key;
                        temp_2 = n*delta_min*2^(o_key-1) -
y_key;
                        temp_3 =
2*((lambda_descr*sigma_key)^2);
                        c_descr_mn = exp(-(temp_1^2 +
temp_2^2)/temp_3) * orient_magn{o_key}{s_key}{1}(m,n);
                        for i=1:n_hist
                            x_i = (i -
(1+n_hist)/2)*2*lambda_descr/n_hist;
                            for j=1:n_hist
                                y_j = (j -
(1+n_hist)/2)*2*lambda_descr/n_hist;
                                temp_4 = abs(x_i-x_mn);
                                temp_5 = abs(y_j-y_mn);
                                if(temp_4 <=
2*lambda_descr/n_hist && ...
                                        temp_5 <=
2*lambda_descr/n_hist)
                                    for k=1:n_ori
                                        temp_6 = abs(2*pi*(k-
1)/n_ori - mod(theta_mn, 2*pi));
                                        if(temp_6 <
2*pi/n_ori)
                                            hist(i, j, k) =
hist(i, j, k) + ...
                                                (1-
n_hist*temp_4/(2*lambda_descr))*...
                                                (1-
n_hist*temp_5/(2*lambda_descr))*...
                                                (1-
n_ori*temp_6/(2*pi))*c_descr_mn;
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end

            count = count + 1;
            for i=1:n_hist
                for j=1:n_hist
                    for k=1:n_ori
                        f{count}((i-1)*n_hist*n_ori+(j-
1)*n_ori+k) = hist(i, j, k);
```

```
                    end
                end
            end

            f_modul = norm(f{count});
            for l=1:n_hist*n_hist*n_ori
                f{count}(l) = min(f{count}(l), 0.2*f_modul);
                f{count}(l) =
min(floor(512*f{count}(l)/f_modul), 255);
            end
            abs_keypoint{count} = [x_key y_key sigma_key
theta_keys{kp}];
        end
    end

    result = cell(3, 1);
    result{1} = f;
    result{2} = abs_keypoint;
    result{3} = count;

    descriptors = result;
end
```

i. Thuật toán so khớp các keypoint (Matching)

- Giả sử $\mathcal{L}^A$ và $\mathcal{L}^B$ là 2 tập descriptor của các điểm đặc trưng tìm được tương ứng từ 2 ảnh u^A và u^B. Với mỗi keypoint descriptor f^a thuộc $\mathcal{L}^A$:

  + Tìm keypoint descriptor f^b thỏa:

$$f^b = \ argmin\ ||f - f^a||_2, \quad f \in \mathcal{L}^B$$

  + Tìm keypoint descriptor $f^{b\prime}$ thỏa:

$$f^b = \ argmin\ ||f - f^a||_2, \quad f \in \mathcal{L}^B \backslash \{f^b\}$$

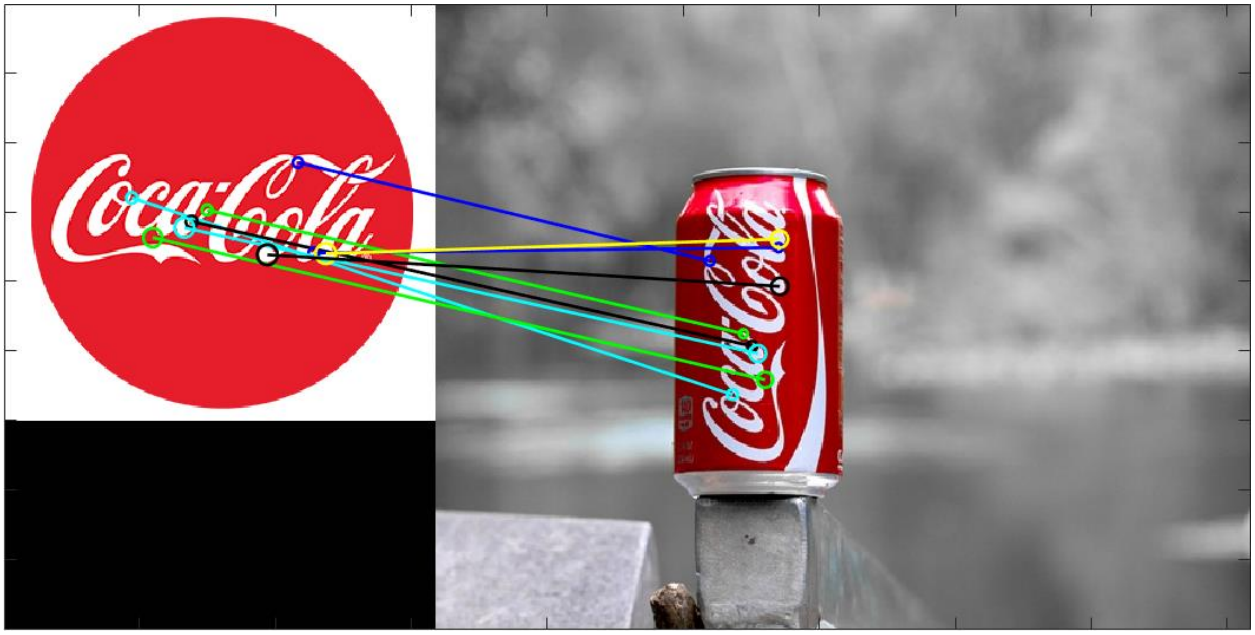
- Nếu $d(f^a, f^b) < C_{relative}^{match} d(f^a, f^{b\prime})$ thì 2 descriptor f^a và f^b là so khớp được với nhau.
- Mã nguồn trên MATLAB:

```
    ...
    num_matches = 0;
    for i=1:descriptors_1{3}
        pos = 0;
        value_min = Inf;
        value_second_min = Inf;
```

```
        for j = 1:descriptors_2{3}
            dist = norm(descriptors_1{1}{i} -
descriptors_2{1}{j});
            if value_second_min > dist
                value_second_min = dist;
            end
            if(value_min > dist)
                pos = j;
                value_second_min = value_min;
                value_min = dist;
            end
        end

        if (value_min < C_match_relative*value_second_min)
            num_matches = num_matches + 1;

plot(descriptors_1{2}{i}(2),descriptors_1{2}{i}(1),Cp{mod(n
um_matches,5)+1},
'MarkerSize',descriptors_1{2}{i}(3)*4+0.5, 'LineWidth', 2);

plot(width_1+descriptors_2{2}{pos}(2),descriptors_2{2}{pos}
(1),Cp{mod(num_matches,5)+1},
'MarkerSize',descriptors_2{2}{pos}(3)*4+0.5, 'LineWidth',
2);

plot([descriptors_1{2}{i}(2),width_1+descriptors_2{2}{pos}(
2)],[descriptors_1{2}{i}(1) descriptors_2{2}{pos}(1)],
C{mod(num_matches,5)+1}, 'LineWidth', 2);
        end
    end

    fprintf('Done matching, num_matches = %d\n',
num_matches);
```

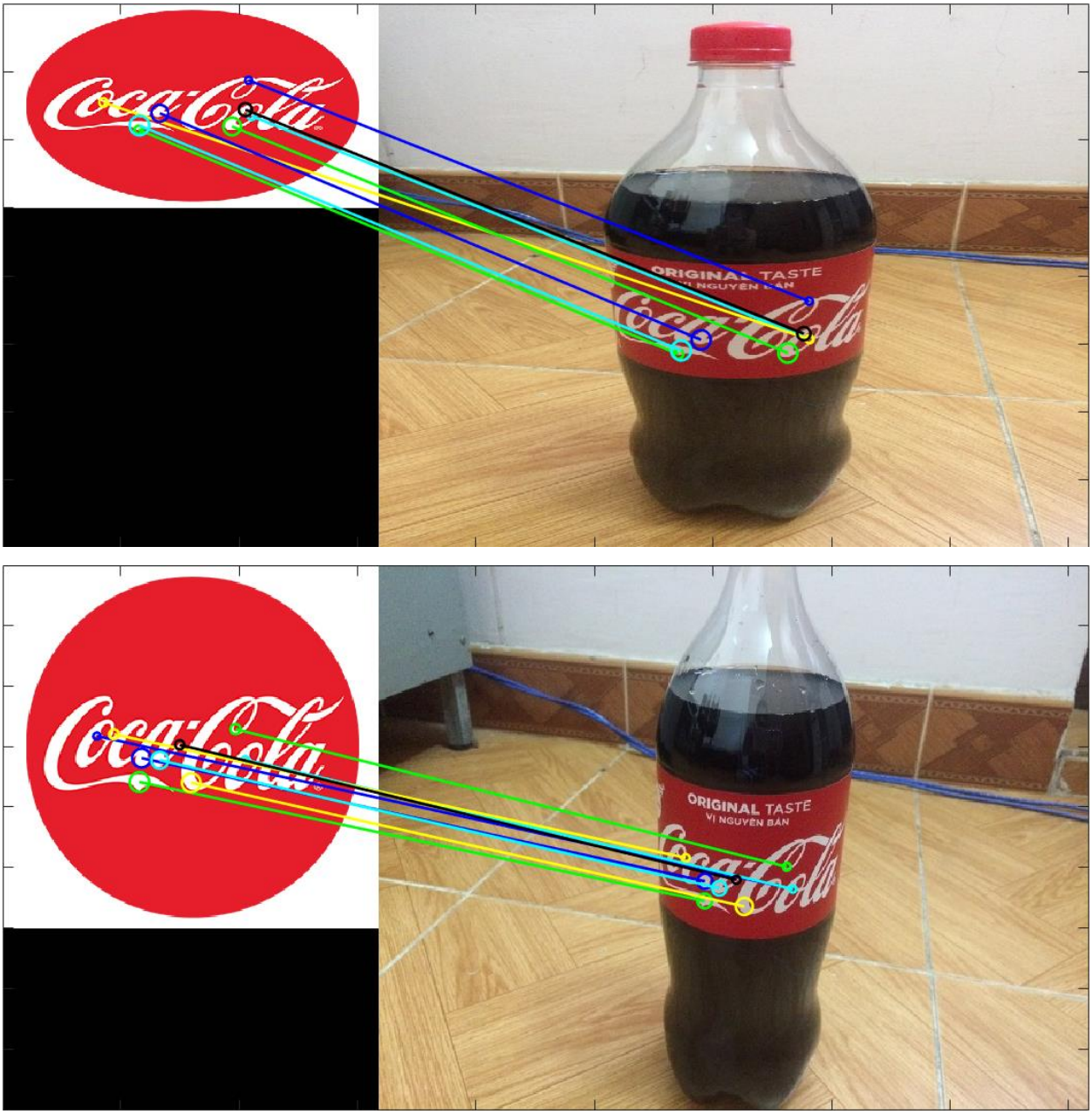
3. KẾT QUẢ THỬ NGHIỆM

- Thử nghiệm với bộ tham số mặc định:



Hình 10: Các kết quả thử nghiệm với bộ tham số mặc định

- Thử nghiệm với ngưỡng so khớp = 0.8:



Hình 11: Các kết quả thử nghiệm với ngưỡng so khớp bằng 0.8, các tham số khác giữ nguyên giá trị mặc định

KẾT LUẬN

Trích chọn các điểm đặc trưng là công việc quan trọng và thường gặp trong các bài toán xử lý ảnh hay thị giác máy tính. Bài báo *"Anatomy of the SIFT Method"* trên IPOL journal đã trình bày và phân tích chi tiết từng bước và các công thức tính toán của thuật toán trích chọn đặc trưng cục bộ phổ biến là SIFT – Scale Invariant Feature Transform. Em đã tiến hành cài đặt theo các bước của thuật toán trên công cụ MATLAB và tiến hành thử nghiệm các kết quả. Thuật toán SIFT mang lại kết quả tốt trong nhiều trường hợp nói chung cũng như trong trường hợp so khớp logo Coca Cola nói riêng. Qua việc tìm hiểu thuật toán trên, em đã tích lũy được cho bản thân ngoài kiến thức các quy trình của thuật toán, còn cả kỹ năng lập trình MATLAB, giải quyết vấn đề trong khi thực hiện. Tuy nhiên do thời gian hạn hẹp nên em chưa thể tiến hành xây dựng một ứng dụng hoàn chỉnh, có thể ứng dụng được trong thực tế từ thuật toán trên. Em rất mong nhận được sự góp ý từ thầy. Em xin chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

1. Ives Rey Otero, Mauricio Delbracio , *Anatomy of the SIFT Method*, 2014, http://www.ipol.im/pub/art/2014/82/

2. Edouard Oyallon, Julien Rabin, An Analysis of the SURF Method, 2015, http://www.ipol.im/pub/art/2015/69/?utm_source=doi