影像處理導論 HW7

宋其諭 0510888

Project goal

Consider the RBG image**, image-pj7a.tif** or **image-pj7b.tif** (either one), construct 400-superpixel image and 100-superpixel image, using threshold *T*=10 and *c* = 1 and 10 (*c*: constant for computing composite distance *D*).

1. Figures of 400-superpixel images for *c*=1 and *c*=10 (30%)

C=1,400 superpixel

C=10,400 superpixel

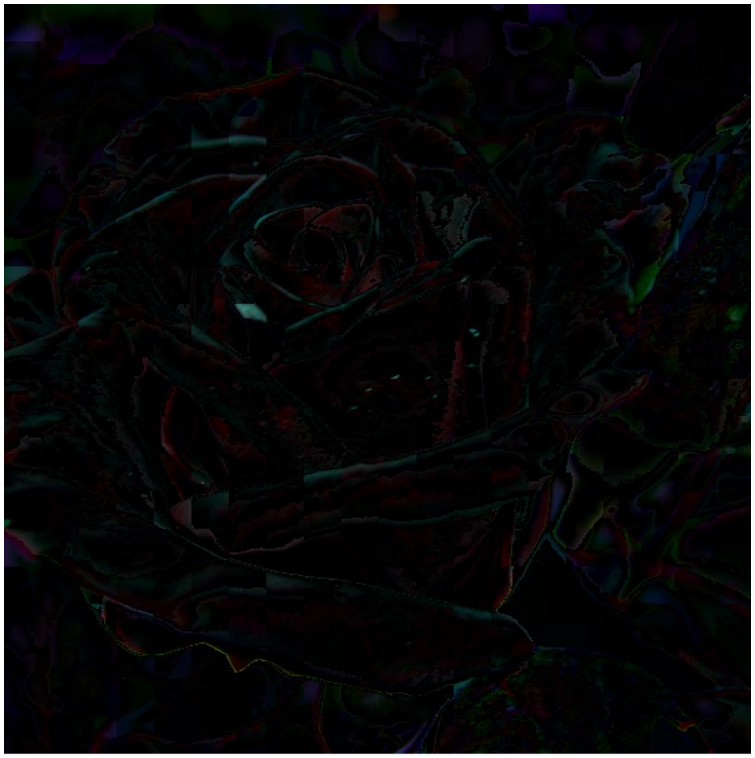
2. Figures of 100-superpixel images for *c*=1 and *c*=10 (20%)

C=1,100 superpixel

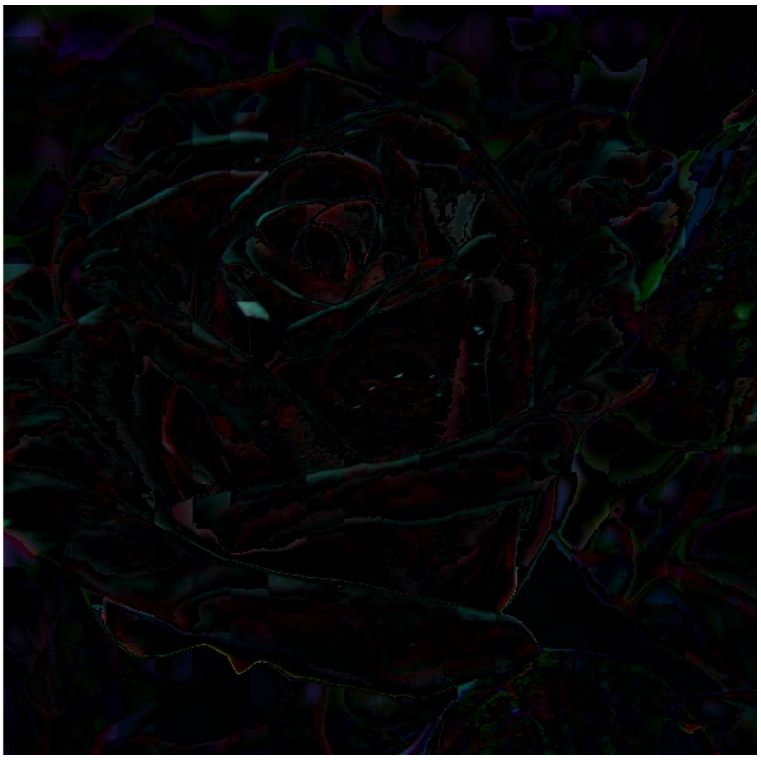
C=10,100 superpixel

3. Difference images between each of the four superpixel image and the original image (20%)

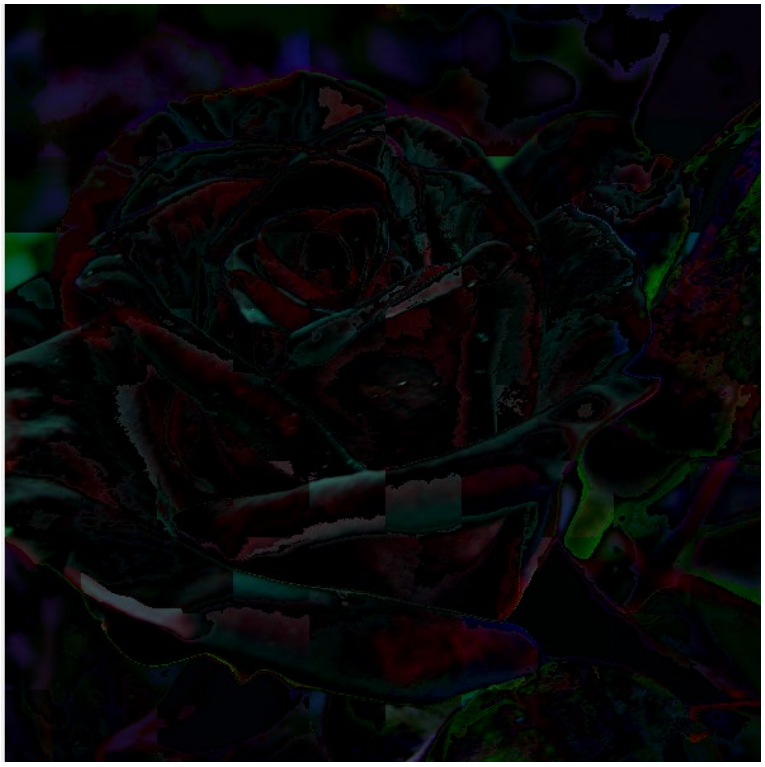
C=1,400 superpixel



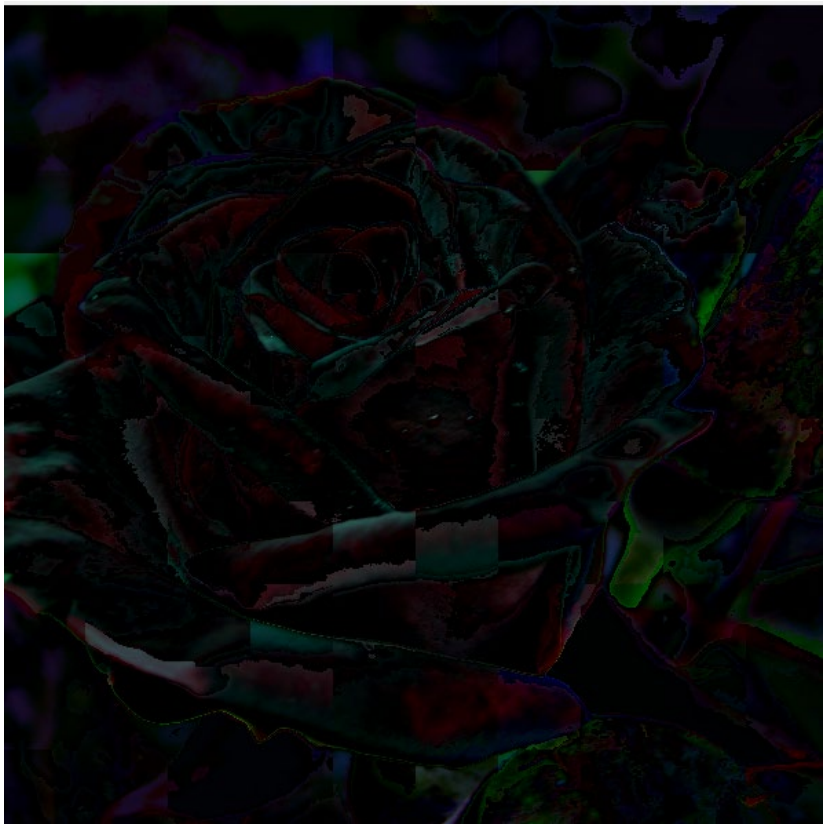
C=10,400 superpixel



C=1,100 superpixel



C=10,100 superpixel



Source codes

本次實驗使用 Matlab 軟體分析(含註解)

```
clear all;   close all;   clc;
img = imread('image-pj7c.tif');
% 600 * 600 * 3
R = double(img(:,:,1));
G = double(img(:,:,2));
B = double(img(:,:,3));
% constant
superpixel = 100;     % 100 & 400
c = 10;

out_img = SLIC(superpixel, c, R, G, B);

figure;
imshow(out_img./255, []);
difference_img = double(img) - out_img;
figure;
imshow(difference_img./255, []);

% subimg = img(49:51, 3:5, :);
function [output] = SLIC(superpixel, c, R, G, B)
    %initial constant
    count = 0;
    threshold = 10;
    L = zeros(600, 600);
    D = zeros(600, 600);
    D = D-1;
    step = floor(sqrt(360000/superpixel));        % s = 60 for 100, s = 30 for 400
    m_seeds = init_seeds(step, superpixel, R, G, B);

    while true
        mi_index = 1;
        for i = step:step:600
            for j = step:step:600
                for k = (i-step+1):(i+step)
                    for l = (j-step+1):(j+step)
                        if((k<601)&&(l<601))
```

```
                    d = calculate_D(m_seeds, mi_index, k, l, R, G, B, step, c);
                    if (d < D(k, l)) || (D(k, l) == -1)
                        D(k, l) = d;
                        L(k, l) = mi_index;
                    end
                end
            end
        end
        mi_index = mi_index + 1;
    end
end

m = zeros(superpixel, 5);
for i = 1:600
    for j = 1:600
        m(L(i, j), 1) = m(L(i, j), 1) + R(i, j);
        m(L(i, j), 2) = m(L(i, j), 2) + G(i, j);
        m(L(i, j), 3) = m(L(i, j), 3) + B(i, j);
        m(L(i, j), 4) = m(L(i, j), 4) + i;
        m(L(i, j), 5) = m(L(i, j), 5) + j;
    end
end
for i = 1:superpixel
    times = length(find(L(:) == i));
    m(i, 1) = m(i, 1)/times;
    m(i, 2) = m(i, 2)/times;
    m(i, 3) = m(i, 3)/times;
    m(i, 4) = floor(m(i, 4)/times)+1;
    m(i, 5) = floor(m(i, 5)/times)+1;
end
for i = 1:superpixel
    L( floor(m(i, 4))+1, floor(m(i, 5))+1 ) = i;
end
% calculate error
error = 0;
E = m_seeds - m;
for i = 1:superpixel
    sum = sqrt(E(i, 1).^2 + E(i, 2).^2 + E(i, 3).^2 + E(i, 4).^2 + E(i, 5).^2);
```

```
            error = error + sum;
        end
        count = count + 1;
        fprintf("%d times, error %f\n", count, error);
        if(error < threshold)
            break
        end
        m_seeds = m;
    end
    fprintf("done");
    for i = 1:600
        for j = 1:600
            output(i, j, 1) = m(L(i, j), 1);
            output(i, j, 2) = m(L(i, j), 2);
            output(i, j, 3) = m(L(i, j), 3);
        end
    end
end

function [m] = init_seeds(step, superpixel, R, G, B)
    m = zeros(superpixel, 5);
    step_sqrt = sqrt(superpixel);
    gradient = find_gradient(R, G, B);
    for i = 1:superpixel
        x = floor(i/step_sqrt)+1;
        y = mod(i, step_sqrt);
        if (y == 0)
            y = step_sqrt;
        end
        if (mod(i, step_sqrt) == 0)
            x = x - 1;
        end
        X = x*step;
        Y = y*step;
        m(i, 1) = R(X, Y);
        m(i, 2) = G(X, Y);
        m(i, 3) = B(X, Y);
        for k = -1:1
```

```
                for l = -1:1
                    displace_x = X+k;
                    displace_y = Y+l;
                    if((displace_x<601)&&(displace_y<601)&&(gradient(X,
Y)>gradient(displace_x, displace_y)))
                        m(i, 4) = displace_x;
                        m(i, 5) = displace_y;
                    else
                        m(i, 4) = X;
                        m(i, 5) = Y;
                    end
                end
            end
        end
end

function [g] = find_gradient(R, G, B)
    [g_r a]= imgradient(R);
    [g_g b]= imgradient(G);
    [g_b c]= imgradient(B);
    g = sqrt(g_r.^2+g_g.^2+g_b.^2);
end

function D = calculate_D(m, mi_index, x, y, R, G, B, s, c)
    Dc = (R(x, y)-m(mi_index, 1)).^2+(G(x, y)-m(mi_index, 2)).^2+(B(x, y)-m(mi_index,
3)).^2;
    Ds = (m(mi_index, 4)-x).^2+(m(mi_index, 5)-y).^2;
    D = sqrt(Dc/c/c+Ds/s/s);
end
```