TOPIC:FOG REMOVER

```
clc;clear all;close all;
fogged = imread("foggy_bench.jpg"); %Fogged Image
figure, imshow(fogged);
```

```
%STEP.1. ESTIMATION OF GLOBAL AIRLIGHT.
frame = 10; % Local Patch Size
for k = 1 : 3
    minimum_intense = ordfilt2(double(fogged(:, :, k)), 1, ones(frame),
'symmetric');
    Airlight(k) = max(minimum_intense(:));
end
```

```
%STEP.2. CALCULATION OF DARK CHANNEL AND APPLY BOUNDARY CONSTRAINT
frame = 3;
```

```
bounded_fogged = boundary_constraint(fogged, Airlight, 30, 300, frame);
figure, imshow(bounded_fogged, []),title('Constraint Bounded Dark Channel Prior
Image');
```


```
%STEP.3. TRANSMISSION MAP CALCULATION.
reg_param = 7;  % regularization parameter.
t = depth_map(fogged, bounded_fogged, reg_param, 0.5);
figure, imshow(1-t, []); colormap hot,title('Transmission Map');
```

Transmission Map

```
%STEP.4. RECOVERY OF DE-FOGGGED IMAGE.
defog_param = 0.85;
t = max(abs(t), 0.0001).^defog_param;
fogged = double(fogged);
if length(Airlight) == 1
    Airlight = Airlight * ones(3, 1);
end
R = (fogged(:, :, 1) - Airlight(1)) ./ t + Airlight(1);
G = (fogged(:, :, 2) - Airlight(2)) ./ t + Airlight(2);
B = (fogged(:, :, 3) - Airlight(3)) ./ t + Airlight(3);
inputImage = cat(3, R, G, B) ./ 255;
figure, imshow(inputImage, []),title('Final Defogged Image');
```

Final Defogged Image

```
i1 = imread("foggy_bench.jpg");
subplot(121);imshow(i1);
subplot(122);imshow(inputImage, []);
```

```
equalizedImage = histeq(inputImage);
figure(); imshow(equalizedImage);
 title('Equalized Image');
```

Equalized Image

```
 sigma=0.5;
 smoothedImage = imgaussfilt(equalizedImage, sigma);
figure(); imshow(smoothedImage);
```

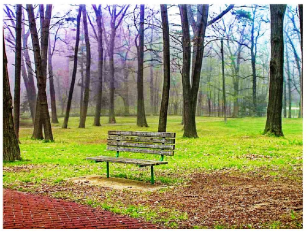
```
subplot(3,2,1);imshow(i1);title('Original image');
subplot(3,2,2);imshow(bounded_fogged),title('Constraint Bounded Dark Channel Prior
Image');
subplot(3,2,3);imshow(1-t, []); colormap hot,title('Transmission Map');
subplot(3,2,4);imshow(inputImage, []),title('Final Defogged Image');
subplot(325);imshow(equalizedImage);
subplot(326);imshow(smoothedImage);
```

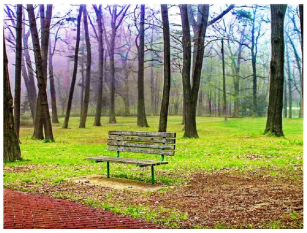
Original image

Constraint Bounded Dark Channel Prior Image

Transmission Map

Final Defogged Image





```
function [p1, p2] = boundary_constraint(fogged, Airlight, C0, C1, frame)
    % patch wise transmission from boundary constraint

    if length(Airlight) == 1
        Airlight = Airlight * ones(3, 1);
    end
    if length(C0) == 1
        C0 = C0 * ones(3, 1);
    end
    if length(C1) == 1
        C1 = C1 * ones(3, 1);
    end
    fogged = double(fogged);

    % pixel-wise boundary
    t_r = max((Airlight(1) - fogged(:, :, 1)) ./ (Airlight(1)  -  C0(1)),
(fogged(:, :, 1) - Airlight(1)) ./ (C1(1) - Airlight(1) ));
    t_g = max((Airlight(2) - fogged(:, :, 2)) ./ (Airlight(2)  - C0(2)),
(fogged(:, :, 2) - Airlight(2)) ./ (C1(2) - Airlight(2) ));
```

```
    p2 = max((Airlight(3) - fogged(:, :, 3)) ./ (Airlight(3)  - C0(3)),
(fogged(:, :, 3) - Airlight(3)) ./ (C1(3) - Airlight(3) ));
    p2 = max(cat(3, t_r, t_g, p2), [], 3);
    p2 = min(p2, 1);

    % minimum filtering
    se = strel('square', frame);
    p1 = imclose(p2, se);
end
function bound_fogged = depth_map(fogged, bound_fogged, reg_param, val)

    [nRows, nCols] = size(bound_fogged);

    % various filters to be applied
    nsz = 3; total = nsz * nsz;
    d{1} = [5, 5, 5; -3, 0, -3; -3, -3, -3];
    d{2} = [-3, 5, 5; -3, 0, 5; -3, -3, -3];
    d{3} = [-3, -3, 5; -3, 0, 5; -3, -3, 5];
    d{4} = [-3, -3, -3; -3, 0, 5; -3, 5, 5];
    d{5} = [5, 5, 5; -3, 0, -3; -3, -3, -3];
    d{6} = [-3, -3, -3; 5, 0, -3; 5, 5, -3];
    d{7} = [5, -3, -3; 5, 0, -3; 5, -3, -3];
    d{8} = [5, 5, -3; 5, 0, -3; -3, -3, -3];

    % normalizing filters
    num_filters = length(d);
    for k = 1 : num_filters
        d{k} = d{k} / norm(d{k}(:));
    end

    % calculating weighting function
    for k = 1 : num_filters
         fogged = double(fogged) / 255;
        d_r = imfilter(fogged(:, :, 1),  d{k}, 'circular');
        d_g = imfilter(fogged(:, :, 2),  d{k}, 'circular');
        d_b = imfilter(fogged(:, :, 3),  d{k}, 'circular');
        Weight = exp(-(d_r.^2 + d_g.^2 + d_b.^2) / val / 2);
        %weight{k} = weight_function(fogged, d{k}, val);
    end

    Tf = fft2(bound_fogged);
    DS = 0;
    for k = 1 : num_filters
        D{k} = psf2otf(d{k}, [nRows, nCols]);
        DS = DS + abs(D{k}).^2;
    end

    xponent = 1; xponent_rate = 2 * sqrt(2);
    xponent_max = 2^8;
```

```
    while xponent < xponent_max

        gamma = reg_param / xponent;

        DU = 0;
        for k = 1 : num_filters
            dt{k} = imfilter(bound_fogged, d{k}, 'circular');
            fogged = double(fogged) / 255;
            d_r = imfilter(fogged(:, :, 1),  d{k}, 'circular');
            d_g = imfilter(fogged(:, :, 2),  d{k}, 'circular');
            d_b = imfilter(fogged(:, :, 3),  d{k}, 'circular');
            Weight = exp(-(d_r.^2 + d_g.^2 + d_b.^2) / val / 2);
            u{k} = max(abs(dt{k}) - Weight / xponent / num_filters, 0) .*
sign(dt{k});
            DU = DU + fft2(imfilter(u{k}, flipud(fliplr(d{k})), 'circular'));
        end

        bound_fogged = abs(ifft2((gamma * Tf + DU) ./ ( gamma + DS)));

        xponent = xponent * xponent_rate;
    end
end
```