PROJECT REPORT

On

Haze Removal Using Dark Channel prior

BY

HARSHUL GUPTA

2018B5A31058H

Under the supervision of

Dr. Sudha Radhika

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

EEE F435 – Digital Image Processing

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(April 2022)

Abstract

In this project, we implemented a simple image prior—dark channel prior to remove haze from a input image. The dark channel prior is a kind of statistical model of outdoor haze-free images. It is based on a key observation—most local patches in outdoor haze-free images contain some pixels whose intensity is very low in at least one color channel. Using this prior with the haze imaging model, we can directly estimate the thickness of the haze and recover a high-quality haze-free image. Results on a variety of hazy images demonstrate the power of the proposed prior. Moreover, a high-quality depth map can also be obtained as a byproduct of haze removal.

INTRODUCTION

I many times came across the outdoor images that generally degraded by presence of several things in the atmosphere. Haze, fog, and smoke are such phenomena due to atmospheric absorption and scattering. The irradiance received by the camera from the scene point is attenuated along the line of sight. Furthermore, the incoming light is blended with the airlight —ambient light reflected into the line of sight by atmospheric particles.As a result images lose contrast and color fidelity. Since the amount of scattering depends on the distance of the scene points from the camera, the degradation is spatially variant. Haze removal(or dehazing) is highly desirable technique in consumer/computational photography and Image processing applications. First, removing haze can significantly increase the visibility of the scene and correct it's color shift caused by the airlight. The haze-free image is more visually pleasing. Second, most computer vision algorithms, from low-level image analysis to high-level object recognition, usually assume that the input image is the scene radiance. The performance of many vision algorithms (e.g., feature detection, filtering, and photometric analysis) will inevitably suffer from the biased and low-contrast scene radiance. Last, haze removal can provide depth information and benefit many vision algorithms and advanced image editing. Haze or fog can be a useful depth clue for scene understanding. We can put A bad hazy image to good use.

Background

In this project, model we used to describe the formation of a haze image is as follows

I(x) = J(x)t(x) + A(1 − t(x))

where I is the observed intensity, J is the scene radiance, A is the global atmospheric light, and t is the medium transmission describing the portion of the light that is not scattered and reaches the camera. The goal of haze removal is to recover J, A, and t from I

The first term J(x)t(x) on the right hand side of Equation is called direct attenuation , and the second term A(1 − t(x)) is called airlight .Direct attenuation describes the scene radiance and its decay in the medium, while airlight results from previously scattered light and leads to the shift of the scene color.

Dark Channel Prior

- The dark channel prior is based on the following observation on haze-free outdoor images: in most of the non-sky patches, at least one color channel has very low intensity at some pixels. In other words, the minimum intensity in such a patch should has a very low value. Formally, for an image J, we define

$$J^{dark}(\mathbf{x}) = \min_{c \in \{r,g,b\}} \left(\min_{\mathbf{y} \in \Omega(\mathbf{x})} (J^c(\mathbf{y})) \right),$$

-
- where Jc is a color channel of J and Ω(x) is a local patch centered at x. Our observation says that except for the sky region, the intensity of J dark is low and tends to be zero, if J is a haze-free outdoor image. We call Jdark the dark channel of J, and we call the above statistical observation or knowledge the dark channel prior

Estimating the Atmospheric Light

- we first assume that the atmospheric light A is given. We will present an automatic method to estimate the atmospheric light
- In most of the previous single image methods, the atmospheric light A is estimated from the most haze-opaque pixel.. But in some cases, the brightest pixel could be on a white car or a white building.
- To avoid that problem We can use the dark channel to improve the atmospheric light estimation. We first pick the top 0.1% brightest pixels in the dark channel. Among these pixels, the pixels with highest intensity in the input image I is selected as the atmospheric light

Estimating the Transmission

- We assume that the transmission in a local patch Ω(x) is constant. We denote the patch's transmission as t ˜(x). Taking the min operation in the local patch on the haze imaging Equation, we have:

$$\tilde{t}(\mathbf{x}) = 1 - \omega \min_{c}(\min_{\mathbf{y}\in\Omega(\mathbf{x})} (\frac{I^{c}(\mathbf{y})}{A^{c}})).$$

-
- The value of ω is application-based. We assumed it to be 0.95.
- In practice, even in clear days the atmosphere is not absolutely free of any particle. So, the haze still exists when we look at distant objects. If we remove the haze thoroughly, the image may seem unnatural and the feeling of depth may be lost. So we can optionally keep a very small amount of haze for the distant objects by introducing a constant parameter ω

Soft Matting

- we apply a soft matting algorithm to refine the transmission.
- Denote the refined transmission map by t(x)
- we minimize the following cost function:
- $$E(\mathbf{t}) = \mathbf{t}^T \mathrm{L} \mathbf{t} + \lambda (\mathbf{t} - \tilde{\mathbf{t}})^T (\mathbf{t} - \tilde{\mathbf{t}}).$$
- L is the Matting Laplacian matrix

Recovering the Scene Radiance

- The final scene radiance J(x) is recovered by:

$$\mathbf{J}(\mathbf{x}) = \frac{\mathbf{I}(\mathbf{x}) - \mathbf{A}}{\max(t(\mathbf{x}), t_0)} + \mathbf{A}.$$

- A typical value of t0 is 0.1.Since the scene radiance is usually not as bright as the atmospheric light, the image after haze removal looks dim. So, we increase the exposure of J(x) for display

Result

Input:-

Output:-

References

K. He, J. Sun and X. Tang, "Single Image Haze Removal Using Dark Channel Prior," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 12, pp. 2341-2353, Dec. 2011, doi: 10.1109/TPAMI.2010.168.

MATLAB CODE

estimateA.m

```
function A = estimateA( I, J, numPixels )
%ESTIMATEA Summary of this function goes here
%   Detailed explanation goes here

    % Make a list of the brightest pixels
    brightestJ = zeros(numPixels,3);
    [x_dim y_dim] = size(J);
    for i = 1:x_dim
        for j = 1:y_dim
            [minElement, index] = min(brightestJ(:,3));
            if J(i,j) > minElement
                brightestJ(index,:) = [i j J(i,j)];
            end
        end
    end
```

```
    % Find the highest intensity pixel from the original
Image using the
    % list calculated above
    highestIntensity = zeros(1,3);
    for i = 1:numPixels
        x = brightestJ(i,1);
        y = brightestJ(i,2);
        intensity = sum(I(x,y,:));
        if intensity > sum(highestIntensity)
            highestIntensity = I(x,y,:);
        end
    end

    % Set as the Atmosphere lighting
    dimI = size(I);
    if numel(dimI) == 3

      A = zeros(x_dim,y_dim,3);

      for a = 1:dimI(3)
            A(:,:,a) = A(:,:,a) + highestIntensity(:,:,a);
      end
    else
      A = zeros(x_dim,y_dim);
      A(:,:) = A(:,:) + highestIntensity(:,:);
    end

end
```

generateLaplacian.m

```
function [ T ] = generateLaplacian( I, T_est)
%GENERATELAPLACIAN2

    dimI = size(I);

    % Taking a box around the pixel
    win_size = 1;
    win_pixels = 9;

    % As per equation in paper when computing the laplacian
    % U is to be added to the window covariance
    U = .000001 ./win_pixels.*eye(3); %eye() return the
identity matrix

    windowIndicies = 1:dimI(1)*dimI(2);
```

```
    windowIndicies = reshape(windowIndicies,dimI(1),dimI(2));

    totalElements = win_pixels^2 * ( dimI(1) - 2 ) * (
dimI(2) - 2 );

    indicies_x = ones(1,totalElements);
    indicies_y = ones(1,totalElements);
    elements = zeros(1,totalElements);

    count = 0;
    for i = (1+win_size):(dimI(2)-win_size)
        for j = (1+win_size):(dimI(1)-win_size)

            % Get the window around i and j
            rangeI = i-win_size:i+win_size;
            rangeJ = j-win_size:j+win_size;
            window = I(rangeJ, rangeI,:);

            % Convert to a vector
            % each column representing a color
            window_vector = reshape(window,win_pixels,3);

            % Calculate the mean and difference
            window_mean = mean(window_vector)';
            diff = window_vector' -
repmat(window_mean,1,win_pixels);

            % both methods of computing covariant produce the
same results
            window_covariance = (diff*diff'/win_pixels)+U;
            %window_covariance =
(window_vector'*window_vector/win_pixels -
window_mean*window_mean')+U;

            % Compute the elements in the L matrix
            % easier to just store these in a spares matrix
            L_element = eye(win_pixels) - (1 + diff' *
inv(window_covariance) * diff) ./ win_pixels;
            L_element = (L_element(:))'; % reshape it to a
single vector

            % Calculate the cordinates in the L matrix that
we are dealing
            % with.  [ coordinates required in a sparse
matrix ]

            % Step 1.  Get the indicies of the current window
```

```
            window_indicies =
reshape(windowIndicies(rangeJ,rangeI),1,win_pixels);

            % Step 2.  Create all combinations of pixels
            x = repmat(window_indicies,win_pixels,1);
            y = x';

            % reformat combination of pixels
            x = (x(:))';
            y = (y(:))';


            indicies_x((count*(win_pixels^2) +
1):(count*(win_pixels^2)+(win_pixels^2))) = x;
            indicies_y((count*(win_pixels^2) +
1):(count*(win_pixels^2)+(win_pixels^2))) = y;
            elements((count*(win_pixels^2) +
1):(count*(win_pixels^2)+(win_pixels^2))) = L_element;

            count = count + 1;
        end
    end

    L =
sparse(indicies_x,indicies_y,elements,dimI(1)*dimI(2),dimI(1)
*dimI(2));
    T = (L + .0001 .* speye(size(L))) \ T_est(:) .* .0001;
    T = reshape(T, size(T_est));
end
```

makeDarkChannel.m

```
function J = makeDarkChannel( I, patch_size )
    % Assuming that this is RGB but overall not requiring it
    [image_x image_y channels] = size(I);
    J = zeros(image_x,image_y);
    tmpPatch =
double(zeros(2*floor(patch_size/2),2*floor(patch_size/2),chan
nels));

    I = padarray(I, [floor(patch_size/2)
floor(patch_size/2)], 'symmetric');

    % I think the size actually returns in order [y x ~ but
doesn't really
    % matter as long as order is kept
```

```
    % padarray resizes the example 300x400 to 314x414.
    % Use original image_x, image_y and add
2*floor(patch_size/2)
    for i = 1:image_x
        minX = i;
        maxX = (i + 2*floor(patch_size/2));
        for j = 1:image_y
            minY = j;
            maxY = (j + 2*floor(patch_size/2));

            % copy all color channels over
            tmpPatch = I(minX:maxX, minY:maxY,:);
            J(i,j) = min(tmpPatch(:)); % find min across all
channels
        end
    end

end
```

subplotImages.m

```
function subplotImages( im1, im2 )

    subplot(1,2,1);
    imshow(im1);
    subplot(1,2,2);
    imshow(im2);

end
```