

Enhancement Techniques for Low-light and Hazy Images

Parth Shah, Suyash Bagad, Saurabh Kolambe

15D070004, 15D070007, 15D070011

Abstract—Images acquired under very low light conditions, where the image features are nearly invisible and the noise is significant, need to be filtered. Similarly, images of outdoor scenes degraded by haze, fog, and smoke due to atmospheric absorption and scattering, naturally need enhancement. In this project, we have implemented several fast enhancement techniques based on de-hazing for single low-light images. The first implementation uses the luminance map to estimate the global atmospheric light and the transmittance owing to the observed similarity between luminance map and dark channel prior (DCP). Another simple but effective method uses DCP to recover a high-quality haze-free image. The former method is shown to have two merits over the latter, firstly, the computational complexity is greatly reduced; and the problem of block artifacts is also addressed.

I. INTRODUCTION

Hazy images are generally covered by mist or haze and thus are not very clear or well defined. Similarly Low light images have lesser dynamic range. Which degrades the image quality and makes it difficult to distinguish or to process such images for applications in computer vision such as object detection, object recognition. Removing haze and enhancing the Low light images can significantly increase the scene visibility and color contrast thus can be helpful in such applications

A. Low light Enhancement

Traditional algorithms for low light enhancement used alpha correction or histogram equalization techniques. Though This techniques is simple to implement but can cause saturation of color and thus sometimes loss of information. Other proposed solutions for this problem used CEM (color estimation model) or enhancement algorithms based on sparse representation of images.

B. Haze removal algorithms

Haze removal problem is ill posed problem if only a single hazy image is input. Many algorithms used multiple images based haze removal. Polarization based method uses multiple hazy input images with different degree of polarization. A Depth based methods used a predefined 3D depth model to filter out the haze. Thus given some prior with the input, Haze removal is comparatively easy. It is been observed that the hazy images have lesser contrast than well defined images. And thus a local contrast maximizing method can also be used.

To enhance single hazy image, a prior has to be derived from the same input image. Dark channel prior can be used for this case. Haze free images tends to have very low pixel value at least in one channel of the (RGB) image. These



Fig. 1: Hazy and Low-light images

dark pixels can be used to directly to estimate the haze transmission in the image. inverted Low light images can show resemblance to a hazy image, thus above dehazing algorithm can be used to enhance the low light images as well. Luminance map is one of the alternative to the dark channel prior which is much faster in computation and gives lesser block artifacts caused by the local patch based algorithm used in Dark channel prior.

II. BACKGROUND

The Goal of enhancing hazy and low light images is:

- Scene restoration
- Depth estimation

A hazy image can be modelled as,

$$I(m, n) = J(m, n) \cdot t(m, n) + A[1 - t(m, n)] \quad (1)$$

- I is the hazy image
- J is the underlying scene radiance
- t is the transmittance
- A is the global atmospheric light

Thus given a hazy input I , we can compute the enhanced image J if t, A are known. The first term $J(x)t(x)$ on the right-hand side is called direct attenuation, and the second term $A(1 - t(x))$ is called airlight. The direct attenuation describes the scene radiance and its decay in the medium, and the airlight results from previously scattered light and leads to the shift of the scene colors. While the direct attenuation is a multiplicative distortion of the scene radiance, the airlight is an additive one. When the atmosphere is homogenous, the transmission t can be expressed as

$$t(x) = 0.25 \times e^{\beta d(x)} \quad (2)$$

where b is the scattering coefficient of the atmosphere and d is the scene depth. This equation indicates that the scene radiance is attenuated exponentially with the depth. If we

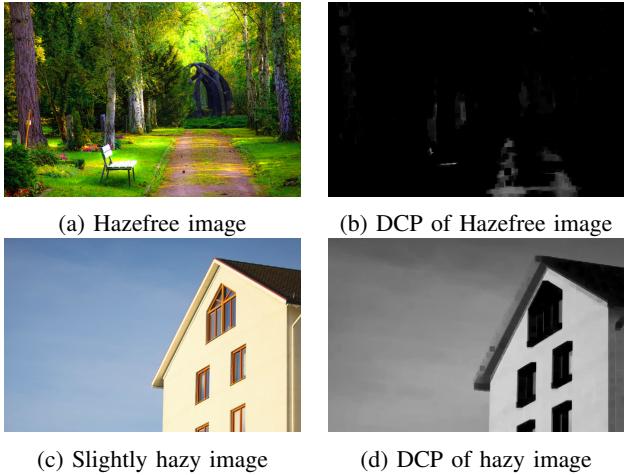


Fig. 2: Dark Channel prior examples

can recover the transmission, we can also recover the depth up to an unknown scale.

The inverted low light images have high similarity with the hazy image and thus can be used modelled using a similar expression,

$$I_{inv}(m, n) = J_{inv}(m, n) \cdot t(m, n) + A[1 - t(m, n)] \quad (3)$$

Where the inverted image is computed as

$$I_{inv} = 255 - I \quad (4)$$

III. METHODS

A. HAZE REMOVAL USING DCP

Firstly, we will demonstrate the haze removal technique using dark channel prior method.

1) Computing DCP: The dark channel prior is based on the observation that the haze-free images has patches with at least one color channel with very low pixel value tending to zero. For a random image, the dark channel prior of the image is given by

$$D^{dark}(x) = \min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (I^c(y))) \quad (5)$$

Intuitively, for computing DCP, we take the minimum of the pixel intensities across all three regions, in a patch Ω . The example of DCP is shown in 2. Due to the additive airlight, a hazy image is brighter than its haze-free version where the transmission t is low. So, the dark channel of a hazy image will have higher intensity in regions with denser haze (as also demonstrated in fig. 2). Visually, the intensity of the dark channel is a rough approximation of the thickness of the haze.

2) Estimating the Atmospheric light: The dark channel of a hazy image approximates the haze denseness (see Fig. 2). So we can use the dark channel to detect the most haze-opaque region and estimate the atmospheric light. We first pick the top 0.1 percent brightest pixels in the dark

channel. These pixels are usually most haze-opaque. We first pick the top 0.1 percent brightest pixels in the dark channel. These pixels are usually most haze-opaque and note that these pixels may not be brightest ones in the whole input image. This simple method based on the dark channel prior is more robust than the "brightest pixel" method. We use it to automatically estimate the atmospheric lights for all images shown in this work.

3) Estimating the Transmission : Now, given that we have the atmospheric light A , we first normalize the haze imaging equation 1 by A :

$$\frac{I^c(x)}{A^c} = t(x) \frac{J^c(x)}{A^c} + 1 - t(x) \quad (6)$$

where each color channel is normalized independently. We further assume that the transmission in a local patch x is constant. We denote this transmission as $\hat{t}(x)$. Now, calculating the dark channel prior on both sides, we have

$$\begin{aligned} & \min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (\frac{I^c(y)}{A^c})) \\ &= 1 - \hat{t}(x) + \min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (\frac{J^c(y)}{A^c})) \hat{t}(x) \end{aligned} \quad (7)$$

Since $\hat{t}(x)$ is a constant in the patch, it can be put on the outside of the min operators. As the scene radiance J is a haze-free image, the dark channel of J is close to zero due to the dark channel prior:

$$J^{dark}(x) = \min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (J^c(y))) = 0 \quad (8)$$

This leads to ($A^c > 0$)

$$\min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (\frac{J^c(y)}{A^c})) = 0 \quad (9)$$

Putting 9 in 7, we get the transmission map as

$$\hat{t}(x) = 1 - \min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (\frac{I^c(y)}{A^c})) \quad (10)$$

This can be written as

$$\hat{t}(x) = 1 - DCP(\frac{I}{A}) \quad (11)$$

In practice, even on clear days the atmosphere is not absolutely free of any particle. So the haze still exists when we look at distant objects. So, we can optionally keep a very small amount of haze for the distant objects by introducing a constant parameter ω ($0 < \omega \leq 1$) into 10 to finally get

$$\boxed{\hat{t}(x) = 1 - \omega \min_{c \in \{R, G, B\}} (\min_{y \in \Omega(x)} (\frac{I^c(y)}{A^c}))} \quad (12)$$



(a) Hazy input image



(b) DCP of the input image



(c) Raw transmission of the input image



(d) Refined transmission of the input image

Fig. 3: Transmission map estimation.

4) Tackling issues of Block artifacts and Patch size: As can be seen from fig. 3a, we could easily see some block artifacts in the transmission estimated directly from the equation 12. These block artifacts depend on the patch size while computing DCP. The dark channel prior becomes better for a larger patch size because the probability that a patch contains a dark pixel is increased, but the assumption that the transmission is constant in a patch becomes less appropriate. If the patch size is too large, halos near depth edges may become stronger. In order to solve this issue, we used a *filter* to refine the raw transmission map. In our implementation, we give the user the choice to select which filter to use among *Guided*, *Bilateral*, *Mean*, *Gaussian*. However, the results for guided filter were by far the best. The reason is that, here we need to filter block artifacts without losing edge information. Thus we need a filter which filters out medievally high frequency components but not the frequency components corresponding to *edges*. The raw and refined transmission map for the image in fig. 3 are shown in 3c, 3d.

5) Recovering Scene Radiance: With the atmospheric light and the transmission map, we can recover the scene radiance easily. But the direct attenuation term $J(x)t(x)$ can be very close to zero when the transmission $t(x)$ is close to zero. The directly recovered scene radiance J is prone to noise. Therefore, we restrict the transmission $t(x)$ by a lower bound t_0 , i.e., we preserve a small amount of haze in very dense haze regions. The final scene radiance $J(x)$ is recovered by

$$J(x) = \frac{I(x) - A}{\max\{t(x), t_0\}} + A \quad (13)$$

A typical value of t_0 is 0.1. However, we have customized it to be given as an input by the user. 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 after recovery.

6) A note on the patch size: A key parameter in this algorithm is the patch size in 12. As discussed already, 9: the larger the patch size, the darker the dark channel. Consequently, 9 is less accurate for a small patch, and the recovered scene radiance is oversaturated. On the other hand, the assumption that the transmission is constant in a patch becomes less appropriate. However, in the results in 4, we have used patch size 9×9 , 15×15 , which shows that our implementation works for sufficiently large patch sizes. This is because the guided filtering technique is able to reduce the artifacts introduced by large patches.

7) Results of Haze removal: We have implemented this very simple but powerful haze removal technique with customization of parameters designed for the user. The results are shown in fig. 4.



Fig. 4: Results of Haze removal implemented on a variety of images (in order, *input image* followed by *reconstructed radiance*.)

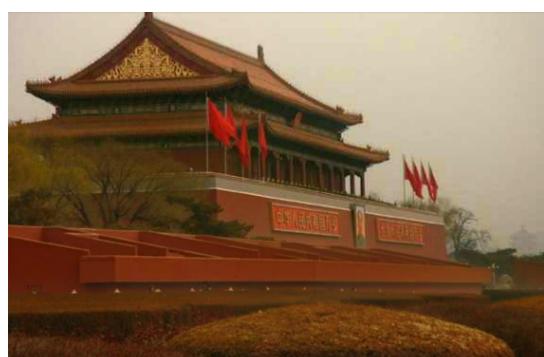
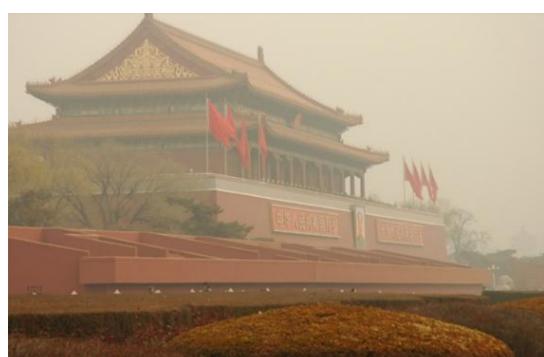


Fig. 5: Results of Haze removal (continued)

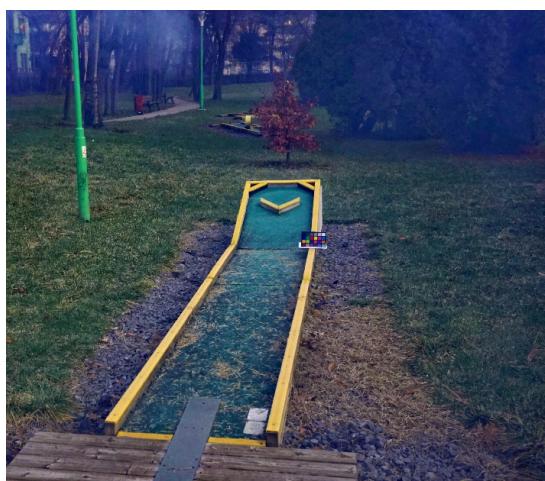


Fig. 7: Results of Haze removal (continued)

Fig. 6: Results of Haze removal (continued)

B. LOW LIGHT IMAGE ENHANCEMENT USING LUMINANCE MAP

In this work, a fast low-light enhancement algorithm is proposed based on de-hazing technique. In our proposed algorithm, luminance map of the inverted low-light image is used to estimate global atmospheric light and transmittance instead of DCP used in previously, based on the analysis of the similarity of the luminance map and the DCP. By doing so, not only the computation complexity can be reduced, but also the block artifacts could be avoided.

This method is inspired by the observation that luminance maps of the inverted low-light images have high similarity with the DCP as shown in Fig. 8. Therefore we will use the luminance map to estimate the global atmospheric light and transmittance instead of the DCP.

Using luminance over DCP has two merits in this proposition. On one hand, the minimum filtering is needed to compute the DCP, which is quite time-consuming; on the other hand, the transmittance in a local patch is assumed to be constant and it is not refined by soft matting because of computation complexity. Thus severe block artifacts would be introduced in the places where transmittance is not continuous. Both of these drawbacks are overcome by this method.

1) Computing Luminance: Instead of DCP, for increasing the computational complexity, luminance map of the inverted low-light image is used to estimate global atmospheric light and transmittance, based on the analysis of the similarity of the luminance map and the DCP. Three color channels of the inverted-low light are weighted summed to computed the luminance map $L(x)$

$$L(x) = 0.299 \times I^R(x) + 0.587 \times I^G(x) + 0.114 \times I^B(x) \quad (14)$$

2) Estimating Atmospheric Light: Considering the similarity between the luminance map and the DCP, we substitute the luminance map for the DCP to estimate the global atmospheric light. The pixel with the highest intensity in the inverted low-light image is selected as the global atmospheric light from the 0.1 % pixels with the highest intensity in the luminance map.

3) Estimating Transmission: Though the inverted low-light image is not a real haze one, its transmittance is closely connected to luminance. The darker the scene is, the denser the corresponding haze of the inverted low-light image. Therefore it is more reasonable to estimate the transmittance using the luminance map. The initial transmittance map $\hat{t}(x)$ is estimated using the luminance map as:

$$\boxed{\hat{t}(x) = 1 - \omega L(x)} \quad (15)$$

where ω is a parameter. The smoother transmittance may allow the underlying scene radiance map to contain more details, so a mean filter is used to obtain the final transmittance $t(x)$:

$$t(x) = \text{meanfilter}(\hat{t}(x)) \quad (16)$$



Fig. 8: Example of DCP and Luminance for a low-light image

The transmittance $t(x)$ tends to be very low, thus the lower bound of the transmittance is limited to $t_0 = 0.01$ in order to avoid being zero as the denominator in the step of recovering the scene radiance. (This too has been implemented in a way that it is user customizable).

4) Recovering Scene Radiance: After obtaining the estimation values of the global atmospheric light and transmittance, the inverted underlying scene radiance $J_{inv}(x)$ can be recovered as

$$\boxed{J_{inv}(x) = \frac{I_{inv}(x) - A}{\max\{t(x), t_0\}} + A} \quad (17)$$

5) Results of Low-light image enhancement: We have implemented this very effective low-light enhancement technique with customization of parameters designed for the user. The results are shown in fig. 9.



Fig. 9: Results of low-light image enhancement implemented on a variety of images (in order, *input image* followed by *reconstructed radiance*.)



Fig. 10: Results of low-light enhancement (continued).



Fig. 11: Results of low-light enhancement (continued).

Fig. 12: Results of low-light enhancement (continued).



Fig. 13: Results of low-light enhancement (continued).

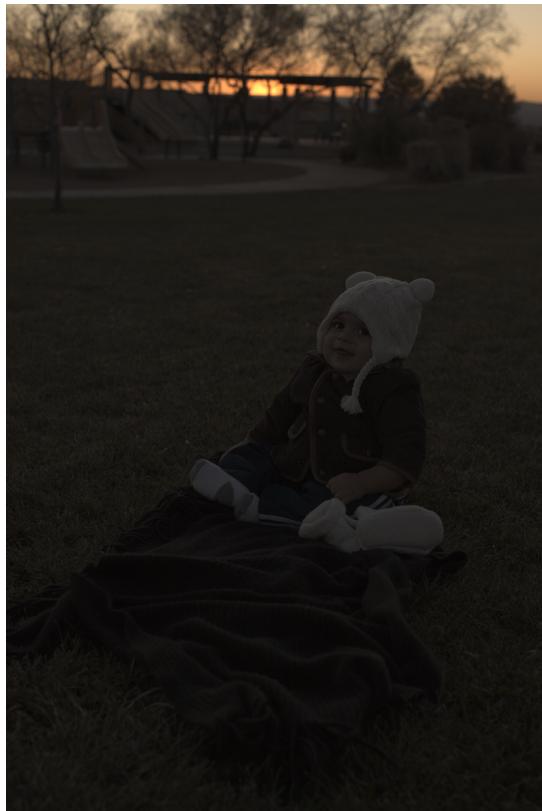


Fig. 14: Results of low-light enhancement, here the transmission map is computed using the luminance of inverted image, but A computed using luminance of original image, leading to superior results.

IV. RESULTS AND CONCLUSION

This work successfully solves the problem of low-light image enhancement and haze removal using very simple but effective techniques.

In the first method, a very simple but powerful prior, called the dark channel prior, is used for single image haze removal. The dark channel prior is based on the statistics of outdoor haze-free images. Combining the prior with the haze imaging model, single image haze removal becomes simpler and more effective.

In the second method, the luminance map of the inverted low-light image is used to estimate the global atmospheric light and transmittance instead of the DCP according to the high similarity between the luminance map and the DCP. Experimental results demonstrate that our proposed algorithm achieves excellent enhancement results in terms of subjective and objective quality. In addition, our proposed algorithm can meet the real-time requirements in practical applications since the computation complexity is greatly reduced without computing DCP using minimum filtering.

The low-light image enhancement method can be used in real-time surveillance applications as the computational speed is much high and could match real-time specifications.

ACKNOWLEDGEMENT

We would personally like to thank Prof. Amit Sethi for designing this project since it really contributed a lot in my learning of this sub-section of the subject. Although it took quite a bit of time to code, but all of it was worth it!

REFERENCES

- [1] Pizer, S.M., Amburn, E.P., Austin, J.D., Cromartie, R., Geselowitz, A., Greer, T., Romeny, B.H., Zimmerman, J.B., Zuiderveld, K.: Adaptive histogram equalization and its variations. *Comput. Vision Graph. Image Process.* 39(3), 355368 (1987)
- [2] Malm, H., Oskarsson, M., Warrant, E., Clarberg, P., Hasselgren, J., Lejdforss, C.: Adaptive enhancement and noise reduction in very low light-level video. In: Proceedings of the 11st IEEE International Conference on Computer Vision, Rio de Janeiro, Brazil, pp. 18, 1421 October 2007
- [3] R. Fattal, Single Image Dehazing, Proc. ACM SIGGRAPH 08, 2008.
- [4] R. Tan, Visibility in Bad Weather from a Single Image, Proc. IEEE Conf. Computer Vision and Pattern Recognition, June 2008.
- [5] M. van Herk, A Fast Algorithm for Local Minimum and Maximum Filters on Rectangular and Octagonal Kernels, *Pattern Recognition Letters*, vol. 13, pp. 517-521, 1992.
- [6] He, K., Sun, J., Tang, X.: Single image haze removal using dark channel prior. *IEEE Trans. Pattern Anal. Mach. Intell.* 33(12), 23412353 (2011)
- [7] Fu, H., Ma, H., Wu, S.: Night removal by color estimation and sparse representation. In: Proceedings of the 21st IEEE International Conference on Pattern Recognition, Tsukuba, Japan, pp. 36563659, 1115 November 2012
- [8] Sen, P., Kalantari, N.K., Yaesoubi, M.: Robust patch-based HDR reconstruction of dynamic scenes. *ACM Trans. Graph.* 31(6), 439445 (2012)
- [9] Goldstein, Amit and Fattal, Raanan, Blur-Kernel Estimation from Spectral Irregularities, European Conference on Computer Vision (ECCV), 2012 622–635.
- [10] A. Hore and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," 2010 20th International Conference on Pattern Recognition, Istanbul, 2010, pp. 2366-2369.
- [11] H. Zhang, W. He, L. Zhang, H. Shen and Q. Yuan, "Hyperspectral Image Restoration Using Low-Rank Matrix Recovery," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 8, pp. 4729-4743, Aug. 2014.
- [12] Juan Song, Liang Zhang, Peiyi Shen, Xilu Peng, and Guangming Zhu, Single Low-Light Image Enhancement Using Luminance Map, *CCPR* 2016, Part II, CCIS 663, pp. 101110, 2016.
- [13] Guo, F., Cai, Z., Xie, B., Tang, J.: Automatic image haze removal based on luminance component. In: Proceedings of the Sixth IEEE International Conference on Wireless Communications Networking and Mobile Computing, Chengdu, China, pp. 14, 2325 September 2010

APPENDIX

The folder of the code and results can be found [here](#). The code can be run on any system with Python 3.3 or newer, along with openCV, numpy, tkinter, PIL, matplotlib packages installed. The implementation is done using a terminal-based GUI, so the steps after running the code `main.py` are self-explanatory. The code of the implementation is:

Return

A scalar 'luminance' value of input image.

```
I_b, I_g, I_r = cv2.split(I)
return (np.array( 0.299 * I_r + 0.587 * I_g
+ 0.114 * I_b )).astype('int')
```

```
def get_dcp(I, h, w):
```

Get the dark channel prior in the (RGB) image data.

Parameters

I: an $M \times N \times 3$ numpy array containing data
([0, L-1]) in the image where
 M is the height, N is the width, 3
represents R/G/B channels.
h: window height
w: window width

Return

An $M \times N$ array for the dark channel prior ($[0, L-1]$).
”””

```

M, N, _ = I.shape
padded = np.pad(I, ((w // 2, w // 2), (h // 2,
h // 2 + (h % 2)), (0, 0)), 'edge')
dark_channel = np.zeros((M, N))
for i, j in np.ndindex(dark_channel.shape):
    dark_channel[i, j] = np.min(padded[i:i + h,
j:j + w, :])
return dark_channel

```

```
def atmospheric_light(I, prior, f):
```

Computes the Global atmospheric light (A) for all three channels of image I.

Parameters

```
I:          the M * N * 3 RGB image data ([0, L-1])
as numpy array
prior:    the prior of the image as an M * N
numpy array
f:          fraction of pixels for estimating the
atmosphere light
```

Return

A 3-element array containing atmosphere light ($[0, L-1]$) for each channel
“””

```
M, N = prior.shape
flatI = I.reshape(M * N , 3)
flatdark = prior.ravel()
```

```
# Find top M * N * f indices
searchidx = (-flatdark).argsort()[:int(M * N * f)]
# print('Atmosphere light region:', [(i // N, i % N) for i in searchidx])
```

```
# Return maximum intensities in each channel
return np.max(flatI.take(searchidx, axis=0), axis=0)
```

```
def transmittance(I, A, h, w, omega=0.95, prior='  
dep', filter_t='guided', tmin=0.15, r=40, eps=1  
e-3):
```

```

111 """
112 Get the transmission esitmate in the (RGB)
113 image data.
114
115 Parameters
116 I: the M * N * 3 RGB image data ([0, L
117 -1]) as numpy array
118 A: a 3-element array containing
119 atmosphere light
120 ([0, L-1]) for each channel
121 h: window height for the estimate
122 w: window width for the estimate
123 omega: bias for the estimate
124 prior: the statistical prior of the image as
125 an M * N numpy array
126 filter: the type of filter to be used for
127 refining raw transmittance
128 tmin: threshold of transmittance
129
130 eps: epsilon for the guided filter
131
132 Return
133 An M * N array containing the transmission rate
134 (transmittance) ([0.0, 1.0])
135 """
136 # Transmittance for DCP based approach
137 if prior == 'dcp':
138     raw_tx = 1.0 - omega * get_dcp(I / A, h, w)
139
140 # Transmittance for luminance based approach
141 elif prior == 'luminance':
142     raw_tx = 1.0 - omega * luminance(I) / 255.0
143
144 # Throw error in any other case
145 else:
146     raise ValueError("The 'prior' argument must
147 either be 'luminance' or 'dcp'.")
148
149 # Refined transmittance based on guided filter
150 if filter_t == 'guided':
151     refined_tx = np.maximum(raw_tx, tmin)
152     normI = (I - I.min()) / (I.max() - I.min())
153     refined_tx = guided_filter(normI,
154     refined_tx, r, eps)
155
156 # Refined transmittance based on bilateral
157 filter
158 elif filter_t == 'bilateral':
159     refined_tx = cv2.bilateralFilter(raw_tx, 9,
160     75, 75)
161     refined_tx = np.maximum(refined_tx, tmin)
162
163 # Refined transmittance based on gaussian
164 filter
165 elif filter_t == 'gaussian':
166     refined_tx = cv2.GaussianBlur(raw_tx, (5,
167     5), 0)
168     refined_tx = np.maximum(refined_tx, tmin)
169
170 # Refined transmittance based on mean filter
171 elif filter_t == 'mean':
172     refined_tx = cv2.blur(raw_tx, (5, 5))
173     refined_tx = np.maximum(refined_tx, tmin)
174
175 else:
176     refined_tx = raw_tx
177
178 return raw_tx, refined_tx
179
180 def get_radiance(I, A, t):
181 """
182 Recover the radiance from raw image data with
183 atmosphere light
184
185 and transmission rate estimate.
186
187 Parameters
188 I: M * N * 3 data as numpy array for the
189 hazy image
190 A: a 3-element array containing atmosphere
191 light
192 ([0, L-1]) for each channel
193 t: estimate of the transmission rate
194
195 Return
196 M * N * 3 numpy array for the recovered
197 radiance
198 """
199 tile_t = np.zeros_like(I)
200 tile_t[:, :, 0] = tile_t[:, :, 1] = tile_t[:, :
201 2] = t
202
203 return ((I - A) / tile_t + A)
204
205 def to_img(raw):
206 """
207 Convert M * N * 3 matrix to 256-bit image data.
208 """
209 # Threshold to [0, 255]
210 cut = np.maximum(np.minimum(raw, 255), 0).
211 astype(np.uint8)
212
213 if len(raw.shape) == 3:
214     b, g, r = cut[:, :, 0], cut[:, :, 1], cut
215     [:, :, 2]
216     R = Image.fromarray(r)
217     G = Image.fromarray(g)
218     B = Image.fromarray(b)
219     cut = Image.merge("RGB", (R, G, B))
220
221 else:
222     return Image.fromarray(cut)
223
224 def dehaze(image, tmin=0.2, Amax=220, h=15, w=15,
225 f=0.0001, omega=0.95, prior='dcp', filter1='
226 guided', r=40, eps=1e-3):
227 """
228 Dehaze the given RGB image.
229
230 Parameters
231 image: the Image object of the RGB image
232 Amax: upper bound of atmospheric light
233 Other parameters same as that of 'transmittance
234 ' function.
235
236 Return
237
238 (dark, rawt, refinedt, rawrad, rerad)
239 Images for dark channel prior, raw transmission
240 estimate,
241 refiend transmission estimate, recovered
242 radiance with raw t,
243 recovered radiance with refined t.
244 """
245 I = np.asarray(image, dtype=np.float64)
246 if prior == 'dcp':
247     Idark = get_dcp(I, h, w)
248 elif prior == 'luminance':
249     Idark = luminance(I)
250
251 else:
252     raise ValueError("The 'prior' argument must
253 either be 'luminance' or 'dcp'.")
254
255 A = atmospheric_light(I, Idark, f)
256 print(A)

```

```

235     raw_t, refined_t = transmittance(I, A, h, w,
236         omega=omega, prior=prior, filter_t = filter1,
237         tmin=tmin, eps=eps)
238     white = np.full_like(Idark, 255)
239
240     return [to_img(raw) for raw in (Idark, white *
241         raw_t, white * refined_t, get_radiance(I, A,
242         raw_t), get_radiance(I, A, refined_t))]

243
244 def bgr_rgb(Ibgr):
245     """
246     Converts BGR image to RGB using Pillow.
247     """
248     b, g, r = Ibgr.split()
249     return Image.merge("RGB", (r, g, b))

250
251 # Main function
252 if __name__ == "__main__":
253     root = Tk()
254     root.withdraw()
255     initialdir="/home/saurabh/Desktop/SEM.7/IP/
256     Project"
257     filename = askopenfilename(initialdir = "/home/
258     saurabh/Desktop/SEM.7/IP/Project", title = "
259     Choose a degraded Image")
260     image = cv2.imread(filename)

261
262     # shutil.rmtree(initialdir+'/output')
263     out_dir = input("Name of the Output Folder : ")
264     os.mkdir(initialdir+"/Output/" + out_dir)
265
266     dest = initialdir+"/Output/" + out_dir + "/"
267
268     image1 = Image.fromarray(image)
269     image1 = bgr_rgb(image1)
270     image1.save(dest + 'input.png')

271     print("Input image read from " + filename)
272     dict_algo = {
273         "d": "DCP-based",
274         "l": "Luminance-based",
275     }

276     while(True):
277         algo = input("Image restoration method:
278             Press d for DCP-based and l for luminance-based
279             .\n")
280         print("Using {0} algorithm".format(
281             dict_algo[algo]))
282         if algo == 'd':
283             # Dehazing using Dark channel prior
284             dark, raw_t, refined_t, raw_rad,
285             refined_rad = dehaze(image, 0.4, 220, 15, 15,
286             0.0001, 0.98, prior='dcp', filter1='guided')

287             # Save results in the output directory
288             dark.save(dest + 'dark_dcp.png')
289             raw_t.save(dest + 'rawt_dcp.png')
290             refined_t.save(dest + 'refinedt_dcp.png')
291
292             raw_rad.save(dest + 'radiance_rawt_dcp.
293             png')
294             refined_rad.save(dest + 'output.png')

295             break
296         elif algo == 'l':
297             # Dehazing using luminance map
298             image_inv = 255 - image
299             Idark, raw_t, refined_t, raw_rad_inv,
300             refined_rad_inv = dehaze(image_inv, 0.2, 220,
301             15, 15, 0.001, 0.95, 'luminance', 'mean')

302
303             # Save results in the output directory
304             white = np.full_like(raw_rad_inv, 255)
305             raw_rad = to_img(white - raw_rad_inv)
306             refined_rad = to_img(white -
307                 refined_rad_inv)

308             b, g, r = raw_rad.split()
309             raw_rad = Image.merge("RGB", (r, g, b))

310             b, g, r = refined_rad.split()
311             refined_rad = Image.merge("RGB", (r, g,
312                 b))

313             Idark.save(dest + 'dark_lum.png')
314             raw_t.save(dest + 'rawt_lum.png')
315             refined_t.save(dest + 'refinedt_lum.png'
316             )
317             raw_rad.save(dest + 'radiance_rawt_lum.
318             png')
319             refined_rad.save(dest + 'output.png')

320             break
321         else:
322             raise ValueError("Invalid input! Try
323                 again.\n")

```

The code for guided filter is:

```

1  #
2  ##########
3  # Enhancement Techniques for Low-Light and Hazy
4  # Images
5  #
6  # Guided filter implementation as a supplement
7  # to
8  # original problem of dehazing using DCP and
9  # luminance.
10 #
11 # Suyash Bagad | Saurabh Kolambe | Parth Shah
12 # 15D070007 | 15D070011 | 15D070004
13 #
14 #
15 # Project: Enhancement Techniques for Low-Light and
16 # Hazy Images
17 # EE 610: Image Processing
18 # Autumn Semester, 2018
19 # Department of Electrical Engineering ,
20 # IIT Bombay
21 # Copyrights reserved @ Suyash Bagad
22 #
23 ##########
24
25 from itertools import combinations_with_replacement
26 from collections import defaultdict
27 import numpy as np
28 from numpy.linalg import inv
29
30 R, G, B = 0, 1, 2
31
32 def boxfilter(I, r):
33     """
34     Fast box filter implementation.
35
36     Parameters
37     -----
38     I: a single channel/gray image data normalized
39         to [0.0, 1.0]
40     r: window radius

```

```

37
38     Return
39     _____
40     The filtered image data.
41     """
42     M, N = I.shape
43     dest = np.zeros((M, N))
44
45     # cumulative sum over Y axis
46     sumY = np.cumsum(I, axis=0)
47     # difference over Y axis
48     dest[:r + 1] = sumY[r: 2 * r + 1]
49     dest[r + 1:M - r] = sumY[2 * r + 1:] - sumY[:M - 2 * r - 1]
50     dest[-r:] = np.tile(sumY[-1], (r, 1)) - sumY[M - 2 * r - 1:M - r - 1]
51
52     # cumulative sum over X axis
53     sumX = np.cumsum(dest, axis=1)
54     # difference over Y axis
55     dest[:, :r + 1] = sumX[:, r:2 * r + 1]
56     dest[:, r + 1:N - r] = sumX[:, 2 * r + 1:] - sumX[:, :N - 2 * r - 1]
57     dest[:, -r:] = np.tile(sumX[:, -1][:, None], (1, r)) -
58         sumX[:, N - 2 * r - 1:N - r - 1]
59
60     return dest
61
62
63 def guided_filter(I, p, r=40, eps=1e-3):
64     """
65     Refine a filter under the guidance of another (RGB) image.
66
67     Parameters
68     _____
69     I: an M * N * 3 RGB image for guidance.
70     p: the M * N filter to be guided
71     r: the radius of the guidance
72     eps: epsilon for the guided filter
73
74     Return
75     _____
76     The guided filter.
77     """
78     M, N = p.shape
79     base = boxfilter(np.ones((M, N)), r)
80
81     # each channel of I filtered with the mean
82     # filter
83     means = [boxfilter(I[:, :, i], r) / base for i
84             in range(3)]
85     # p filtered with the mean filter
86     mean_p = boxfilter(p, r) / base
87     # filter I with p then filter it with the mean
88     # filter
89     means_IP = [boxfilter(I[:, :, i] * p, r) / base
90                 for i in range(3)]
91     # covariance of (I, p) in each local patch
92     covIP = [means_IP[i] - means[i] * mean_p for i
93             in range(3)]
94
95     # variance of I in each local patch: the matrix
96     # Sigma in ECCV10 eq.14
97     var = defaultdict(dict)
98     for i, j in combinations_with_replacement(range(3), 2):
99         var[i][j] = boxfilter(
100             I[:, :, i] * I[:, :, j], r) / base -
101             means[i] * means[j]
102
103     a = np.zeros((M, N, 3))
104     for y, x in np.ndindex(M, N):
105         # rr, rg, rb

```

```

99         # Sigma = rg, gg, gb
100        #           rb, gb, bb
101        Sigma = np.array([[var[R][R][y, x], var[R][G][y, x],
102                           var[R][B][y, x]], [var[R][G][y, x], var[G][
103                               G][y, x], var[G][B][y, x]], [var[R][B][y, x], var[G][
104                               B][y, x], var[B][B][y, x]]])
105        cov = np.array([c[y, x] for c in covIP])
106        a[y, x] = np.dot(cov, inv(Sigma + eps * np.
107            eye(3)))
108
109        b = mean_p - a[:, :, R] * means[R] - \
110            a[:, :, G] * means[G] - a[:, :, B] * means[B]
111
112        q = (boxfilter(a[:, :, R], r) * I[:, :, R] +
113              boxfilter(a[:, :, G], r) *
114                  I[:, :, G] + boxfilter(a[:, :, B], r) * I
115                 [:, :, B] + boxfilter(b, r)) / base
116
117    return q

```