import cv2

import numpy as np

%matplotlib inline

from matplotlib import pyplot as plt

from google.colab.patches import cv2\_imshow

from itertools import combinations\_with\_replacement

from collections import defaultdict

import numpy as np

from numpy.linalg import inv

def get\_illumination\_channel(I, w):

M, N, \_ = I.shape

padded = np.pad(I, ((int(w/2), int(w/2)), (int(w/2), int(w/2)), (0, 0)), 'edge')

darkch = np.zeros((M, N))

brightch = np.zeros((M, N))

for i, j in np.ndindex(darkch.shape):

darkch[i, j] = np.min(padded[i:i + w, j:j + w, :])

brightch[i, j] = np.max(padded[i:i + w, j:j + w, :])

return darkch, brightch

def get\_atmosphere(I, brightch, p=0.1):

M, N = brightch.shape

flatI = I.reshape(M\*N, 3)

flatbright = brightch.ravel()

searchidx = (-flatbright).argsort()[:int(M\*N\*p)]

A = np.mean(flatI.take(searchidx, axis=0), dtype=np.float64, axis=0)

return A

def get\_initial\_transmission(A, brightch):

A\_c = np.max(A)

init\_t = (brightch-A\_c)/(1.-A\_c)

return (init\_t - np.min(init\_t))/(np.max(init\_t) - np.min(init\_t))

def get\_corrected\_transmission(I, A, darkch, brightch, init\_t, alpha, omega, w):

im3 = np.empty(I.shape, I.dtype);

for ind in range(0, 3):

im3[:, :, ind] = I[:, :, ind] / A[ind]

dark\_c, \_ = get\_illumination\_channel(im3, w)

dark\_t = 1 - omega\*dark\_c

corrected\_t = init\_t

diffch = brightch - darkch

for i in range(diffch.shape[0]):

for j in range(diffch.shape[1]):

if(diffch[i, j] < alpha):

corrected\_t[i, j] = dark\_t[i, j] \* init\_t[i, j]

return np.abs(corrected\_t)

def get\_final\_image(I, A, refined\_t, tmin):

refined\_t\_broadcasted = np.broadcast\_to(refined\_t[:, :, None], (refined\_t.shape[0], refined\_t.shape[1], 3))

J = (I-A) / (np.where(refined\_t\_broadcasted < tmin, tmin, refined\_t\_broadcasted)) + A

return (J - np.min(J))/(np.max(J) - np.min(J))

def dehaze(I, tmin, w, alpha, omega, p, eps, reduce=False):

m, n, \_ = I.shape

Idark, Ibright = get\_illumination\_channel(I, w)

A = get\_atmosphere(I, Ibright, p)

init\_t = get\_initial\_transmission(A, Ibright)

if reduce:

init\_t = reduce\_init\_t(init\_t)

corrected\_t = get\_corrected\_transmission(I, A, Idark, Ibright, init\_t, alpha, omega, w)

normI = (I - I.min()) / (I.max() - I.min())

refined\_t = guided\_filter(normI, corrected\_t, w, eps)

J\_refined = get\_final\_image(I, A, refined\_t, tmin)

enhanced = (J\_refined\*255).astype(np.uint8)

f\_enhanced = cv2.detailEnhance(enhanced, sigma\_s=10, sigma\_r=0.15)

f\_enhanced = cv2.edgePreservingFilter(f\_enhanced, flags=1, sigma\_s=64, sigma\_r=0.2)

return f\_enhanced

def reduce\_init\_t(init\_t):

init\_t = (init\_t\*255).astype(np.uint8)

xp = [0, 32, 255]

fp = [0, 32, 48]

x = np.arange(256)

table = np.interp(x, xp, fp).astype('uint8')

init\_t = cv2.LUT(init\_t, table)

init\_t = init\_t.astype(np.float64)/255

return init\_t

im = cv2.imread('dark.png')

plt.figure(figsize=(10,18))

orig = im.copy()

tmin = 0.1 # minimum value for t to make J image

w = 15 # window size, which determine the corseness of prior images

alpha = 0.4 # threshold for transmission correction

omega = 0.75 # this is for dark channel prior

p = 0.1 # percentage to consider for atmosphere

eps = 1e-3 # for J image

I = np.asarray(im, dtype=np.float64) # Convert the input to an array.

I = I[:, :, :3] / 255

f\_enhanced = dehaze(I, tmin, w, alpha, omega, p, eps)

f\_enhanced2 = dehaze(I, tmin, w, alpha, omega, p, eps, True)

cv2\_imshow(orig)

cv2\_imshow(f\_enhanced)

cv2\_imshow(f\_enhanced2)

cv2.waitKey(0)

cv2.destroyAllWindows()

