Haze removal using Dark Channel Prior and Guided Filter

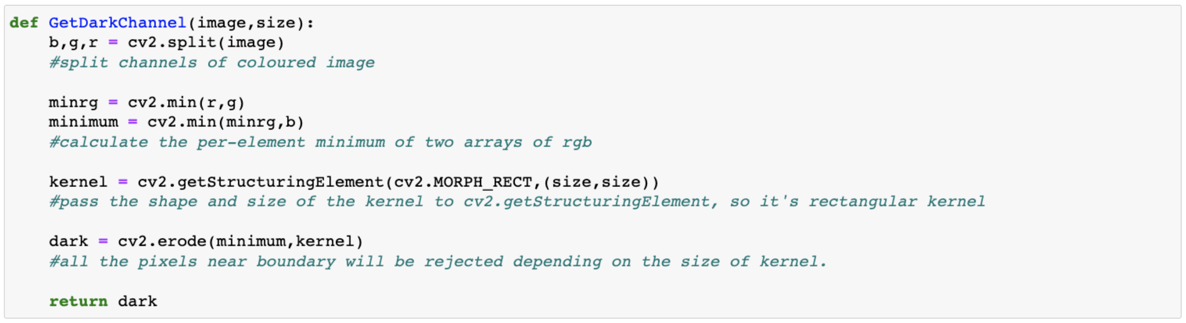
1. Theoretical part

Introductory definition:
Prior - assumption that was built earlier
Dark pixels - when a pixel's intensity is very low or near zero in at least one RGB color channel (a bright green pixel can be a dark pixel, if it's red/blue component is dark)

Conception of **Dark Channel Prior**:
Find the dark pixels (minimum) of each pixel from the RGB channel.

Mathematically: $$\min_{c\in\{r,g,b\}}\Big(\min_{\mathbf{x}'\in\Omega(\mathbf{x})} J^c(\mathbf{x}')\Big) \approx 0, \qquad (1.1)$$

I'm going to use Dark Channel Prior to construct a single image haze removal algorithm.



```
def GetDarkChannel(image,size):
    b,g,r = cv2.split(image)
    #split channels of coloured image

    minrg = cv2.min(r,g)
    minimum = cv2.min(minrg,b)
    #calculate the per-element minimum of two arrays of rgb

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(size,size))
    #pass the shape and size of the kernel to cv2.getStructuringElement, so it's rectangular kernel

    dark = cv2.erode(minimum,kernel)
    #all the pixels near boundary will be rejected depending on the size of kernel.

    return dark
```

Haze imaging equation:

$$I(x) = J(x)t(x) + A(1 - t(x)) \qquad (1.2)$$

- x = (x, y) of a pixel's position in the image.
- I is the hazy image observed. I(x) is a 3D RGB vector of the color at a pixel.
- J represents the scene radiance image (refer to haze free image)
- t is a map called transmission or transparency of the haze. t(x) is a scalar in [0, 1]. Intuitively, t(x) = 0 means completely hazy and opaque, t(x) = 1 means haze-free and completely clear, and 0 < t(x) < 1 means semi-transparent.

- A is the atmospheric light. It is a 3D RGB vector usually assumed to be spatially constant. It is often considered as “the color of the atmosphere, horizon, or sky”

So, if we consider a hazy image’s dark channel, it’s not dark, it’s brightened by the airlight. So we have to solve this problem by using the haze imaging equation and Dark Channel Prior, and estimate the **transmission** and atmospheric light.
If we normalize the equation (1.2), we get the transmission as :

$$\min_{\mathbf{x}'\in\Omega(\mathbf{x})} \min_c \frac{I_c(\mathbf{x}')}{A_c} = \tilde{t}(\mathbf{x}) \min_{\mathbf{x}'\in\Omega(\mathbf{x})} \min_c \left(\frac{J_c(\mathbf{x}')}{A_c}\right) + 1 - \tilde{t}(\mathbf{x}). \qquad (1.3)$$

The scene radiance J is the haze free image, so we have:

$$J^{\text{dark}}(\mathbf{x}) \equiv \min_{\mathbf{x}'\in\Omega(\mathbf{x})} \min_c J_c(\mathbf{x}') \approx 0. \qquad (1.4)$$

It leads to:

$$\tilde{t}(\mathbf{x}) = 1 - \min_{\mathbf{x}'\in\Omega(\mathbf{x})} \min_c \frac{I_c(\mathbf{x}')}{A_c}. \qquad (1.5)$$

The formula 1.5 is the main formula for our transmission.

```python
def GetTransmEstimate(image,A,size):
    newim = np.empty(image.shape,image.dtype)
    #get a new copy of a image with the same shape and datatype

    for each in range(0,3):
        newim[:,:,each] = image[:,:,each]/A[0,each]
    #transfer it to grayish image, bcs in grayish image Ar = Ag = Ab

    omega = 0.95
    # omega is fraction of haze to keep in image (default is 0.95)
    transmission = 1 - omega*GetDarkChannel(newim,size)
    # t = 1-darkchannel/atmosphericlight

    return transmission
```

Then let’s move to the **atmospheric light**.
Let’s assume that the color of the atmosphere is grayish, it means that each component of this color in the RGB channel is the same. Then by using the transmission formula (1.5), we know that :

$$\begin{aligned}\tilde{t}(\mathbf{x}) &= 1 - \frac{1}{A} \min_{\mathbf{x}'\in\Omega(\mathbf{x})} \min_c I_c(\mathbf{x}') \\ &= 1 - \frac{1}{A} I^{\text{dark}},\end{aligned} \qquad (1.6)$$

As the minimum t(x) means the most hazy region, so the maximum(brightest) I^{dark} gives us the minimum t(x). So we use it for the detection of the atmospheric light for pictures.

```python
def GetAtmosphericLight(image,dark):
    [h,w] = image.shape[:2]
    imagesize = h*w
    #image.shape returns (height, width, channels) we took the height, width

    numberofpixels = int(max(math.floor(imagesize/1000),1))


    vec_dark = dark.reshape(imagesize)
    vec_image = image.reshape(imagesize,3)
    #reshape the dark image and original image

    ids = vec_dark.argsort()
    #sort the vectors and return indices of them
    ids = ids[imagesize-numberofpixels::]

    atmos = np.zeros([1,3])
    #create an array to store each window value

    for each in range(1,numberofpixels):
        atmos = atmos + vec_image[ids[each]]

    A = atmos / numberofpixels

    return A
```

Now we have both transmission and atmospheric light, so we can find the **scene radiance J** by:

$$J_c(\mathbf{x}) = \frac{I_c(\mathbf{x}) - A_c}{t(\mathbf{x})} + A_c. \qquad (1.7)$$

But in real practice, the value of t(x) is often very close to zero, so by using the equation (1.7), we might get a noisy scene radiance. So we have to fix t(x) by a lower bound t_0 so we can keep a small amount of haze in very hazy regions. So we obtain scene radiance J as:

$$J_c(\mathbf{x}) = \frac{I_c(\mathbf{x}) - A_c}{\max(t(\mathbf{x}), t_0)} + A_c. \qquad (1.8)$$

```python
def Refine(image,et):
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    #convert image's colorspace to gray
    gray = np.float64(gray)/255
    #grayscale the image
    r = 60
    eps = 0.0001
    t = GetGuidedfilter(gray,et,r,eps)
    return t
```

```python
def Recover(image,t,A,tx = 0.1):
    newim = np.empty(image.shape,image.dtype)
    #copy the image
    t = cv2.max(t,tx)
    #check the transmission's lower bound
    for each in range(0,3):
        newim[:,:,each] = (image[:,:,each]-A[0,each])/t + A[0,each]

    return newim
```

Guided filter:

Guided filter computes the local optimal for each window, we assume that q is a linear transform of I in a window wk centered at the pixel k:

$$q_i = a_k I_i + b_k, \forall i \in w_k, \qquad (1.9)$$

And we hope that p(input) and q(output) have similar colors, so we have to minimize this cost function:

$$\begin{aligned} E(a_k, b_k) &= \sum_{i \in w_k} \left((p_i - q_i)^2 + \epsilon a_k^2 \right) \\ &= \sum_{i \in w_k} \left((p_i - a_k I_i - b_k)^2 + \epsilon a_k^2 \right). \end{aligned} \qquad (1.10)$$

The solution for (1.10) can be given as:

$$\begin{aligned} a_k &= \frac{\operatorname{cov}_k(I,p)}{\sigma_k^2 + \epsilon} \\ b_k &= \bar{p}_k - a_k \mu_k. \end{aligned} \qquad (1.11)$$

Here, μ_k and σ^2_k are the mean and variance of I in w_k, p_k is the mean of p in w_k, and $cov_k(I,p)$ is the covariance of I and p inside the window k. So the linear model for each window is:

$q_i = a_k I_i + b_k$ (1.12)

After computing for all windows:

$$\begin{aligned} q_i &= \frac{1}{|w|} \sum_{k: i \in w_k} (a_k I_i + b_k) \\ &= \bar{a}_i I_i + \bar{b}_i \end{aligned} \qquad (1.13)$$

So the steps of guided filter algorithm is:

1.
$$a_k := \frac{\operatorname{cov}_k(I,p)}{\sigma_k^2 + \epsilon}$$

2.
$$b_k := \bar{p}_k - a_k \mu_k$$

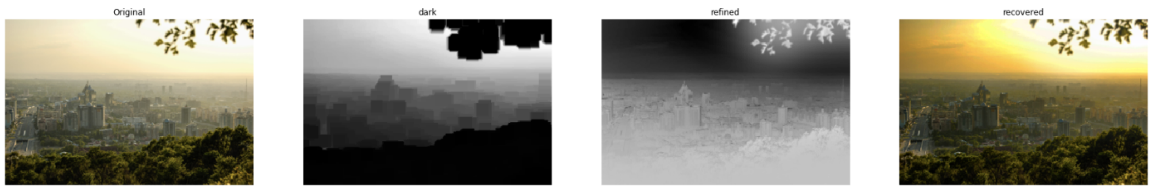
3.
$$q_i := \frac{1}{|w|} \left((\sum_{k \in w_i} a_k) I_i + (\sum_{k \in w_i} b_k) \right) \qquad (1.14)$$

```python
def GetGuidedfilter(image,p,r,eps):
    imean = cv2.boxFilter(image,cv2.CV_64F,(r,r))
    pmean = cv2.boxFilter(p, cv2.CV_64F,(r,r))
    ipmean = cv2.boxFilter(image*p,cv2.CV_64F,(r,r))
    #avreag and bluring
    #why cv2.CV_64F? because it's always safer to take bigger sizes

    ipcov = ipmean - imean*pmean
    #coveriance

    iimean = cv2.boxFilter(image*image,cv2.CV_64F,(r,r))
    ivar   = iimean - imean*imean

    a = ipcov/(ivar + eps)
    b = pmean - a*imean

    amean = cv2.boxFilter(a,cv2.CV_64F,(r,r))
    bmean = cv2.boxFilter(b,cv2.CV_64F,(r,r))

    q = amean*image + bmean
    return q
```

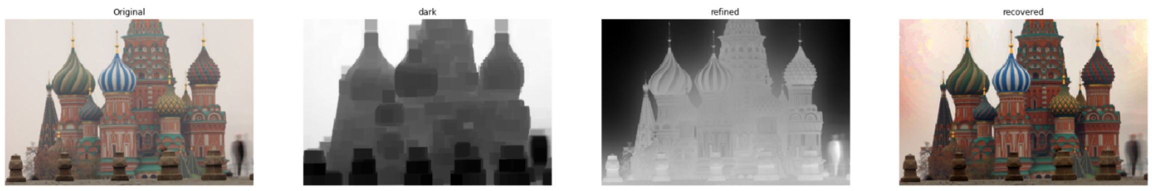
Results:

```
fn = 'Almaty Small.png'
src = cv2.imread(fn)
rgb = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)
I = rgb.astype('float64')/255
dark = GetDarkChannel(I,15)
A = GetAtmosphericLight(I,dark)
te = GetTransmEstimate(I,A,15)
t = Refine(src,te)
J = Recover(I,t,A,0.1)
f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(30,30))
gray = cv2.cvtColor(src,cv2.COLOR_BGR2RGB)
ax1.imshow(gray)
ax1.set_title("Original")
ax1.axis("off")
ax2.imshow(dark, cmap="gray")
ax2.set_title("dark")
ax2.axis("off")
ax3.imshow(t, cmap = "gray")
ax3.set_title("refined")
ax3.axis("off")
ax4.imshow(J,plt.cm.Spectral)
ax4.set_title("recovered")
ax4.axis("off")
plt.show()
```

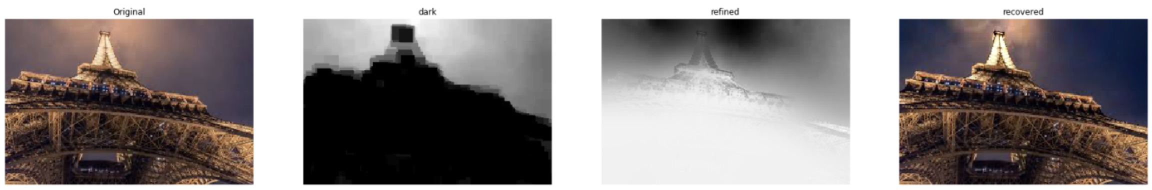
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



(picture of city Almaty, Kazakhstan)



(picture of Red Square, Moscow, Russia)



(picture of Eiffel Tower, Paris, France)

References:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.672.3815&rep=rep1&type=pdf
http://kaiminghe.com/publications/eccv10guidedfilter.pdf