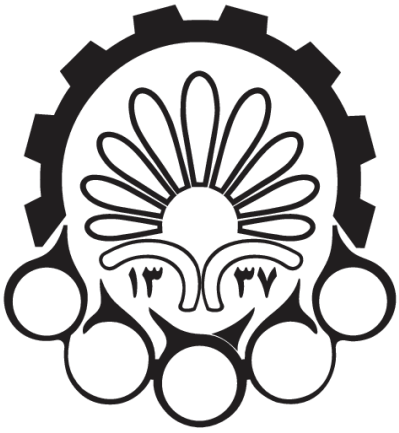


دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

ارائه پیاده سازی پروژه پردازش تصویر

موضوع: **حذف مه از تصویر**

تابستان ۴۰۳

علیرضا دلاوری

Alireza.delavari130@gmail.com
Alirezadelavari@aut.ac.ir
شماره دانشجویی: ۴۰۲۱۳۱۰۲۰

فهرست مطالب

فهرست اشکال

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

بخش اول: مقدمه

در این گزارش به بررسی چگونگی پیاده‌سازی روش بر پایه رتنیکس و هرم لاپلاسین(Retnes-Based Laplacian Pyramid Method for Image Defogging) برای حذف مه از تصویر می‌پردازیم.

روند مه‌زدایی با استفاده از این روش به ۴ بخش کلی تقسیم می‌شوند که عبارت‌اند از: تخمین روشنایی تصویر(illumination) با استفاده از MSRCR، نویززدایی انعکاس(reflection) با استفاده از فیلتر دوطرفه، بهبود جزئیات انعکاس با استفاده از هرم لاپلاسین و بهبود رنگ روشنایی تصویر با استفاده از اصلاح گاما(gamma correction).

در ابتدا، به منظور استخراج بیشتر جزئیات از تصویر(؟)، تصویر ورودی را با ۳ کرنل مختلف گوسین(با اندازه ۳*۳) کانوالو می‌کنیم. سپس با استفاده از MSRCR، تصویر را به بخش انعکاس(reflection) و روشنایی(illumination) تجزیه می‌کنیم. سپس اصلاح گاما بر روی بخش روشنایی تصویر، با پارامترهایی که بر اساس میانگین مقادیر پیکسل های تصویر است به دست می‌آید، انجام می‌شود تا بتوانیم رنگ های اصلی تصویر را در تصویر مه‌زدایی شده باز گردانیم.

همانطور که می‌دانیم بخش انعکاس تصویر که توسط MSRCR استخراج شده است شامل اطلاعات فرکانس بالای تصویر است که می‌تواند شامل لبه ها و نویز ها باشد. روند MSRCR باعث می‌شود که این نویز ها تشدید شوند پس از فیلتر دوطرفه برای نویززدایی بر روی انعکاس تصویر استفاده می‌کنیم.

تصویر حاصل از فیلتر دوطرفه و همچنین تصویر مستقیم انعکاس خروجی از MSRCR، با هم به عنوان ورودی هرم لاپلاسین به کار گرفته می‌شوند.

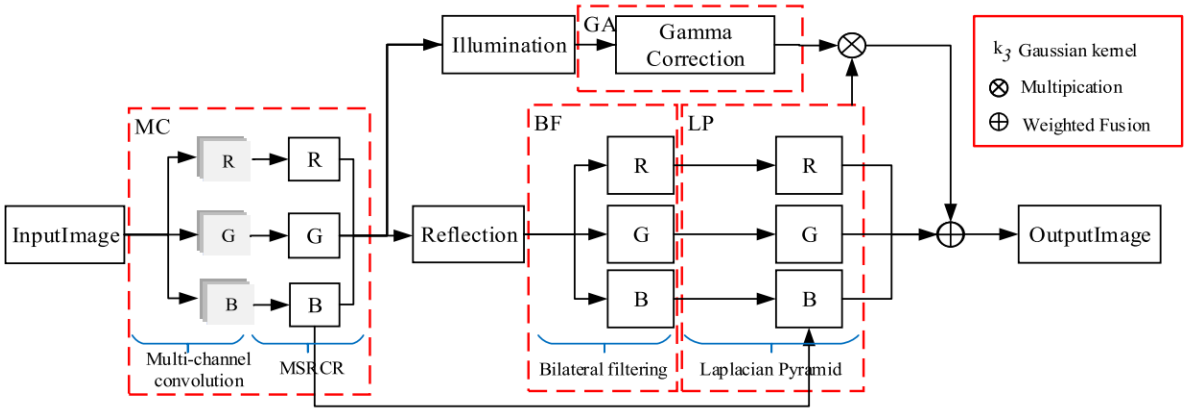
و در انتها برای حفظ طبیعی بودن تصویر، خروجی اصلاح گاما که از قسمت روشنایی تصویر به دست آمده است را با خروجی هرم لاپلاسین به صورت وزن دار خطی ادغام می‌شوند تا تصویر مه‌زدایی شده نهایی را تولید کنند.

بخش دوم: توضیحات پیاده‌سازی

برای مقایسه خروجی هر بخش از تصویر ورودی همراه با مه زیر استفاده شده است:

شکل ۱ تصویر آزمایشی ورودی همراه با مه برای اعتبار سنجی مراحل مختلف

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰



شکل ۲ فلوچارت نشان دهنده مراحل کلی الگوریتم مقاله

شکل بالا فلوچارت الگوریتم را نشان می‌دهد در ادامه به توضیح هر بخش از شکل بالا(مستطیل های قرمز) می‌پردازیم.

MC:

از آنجایی که تئوری این بخش در گزارش فنی به تفصیل توضیح داده شده است فقط به توضیح معادلاتی که در پیاده‌سازی استفاده شده‌اند می‌پردازیم.

روشنایی تصویر را می‌توان با کانوالو کردن تصویر با استفاده از کرنل گوسی تخمین زد. به دلیل زمان محاسبات و مقدار بهبود این کرنل، معمولا از ۳ کرنل گوسی استفاده می‌شود.

$$L_i(x,y) = S_i(x,y) * G_k(x,y)$$

$$G_k(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{(x^2+y^2)}{2\sigma^2}}$$

که در این معادله S تصویر مشاهده شده، L روشنایی است که مربوط به اطلاعات فرکانس های پایین تصویر است و G فیلتر گوسی می‌باشد. این بخش را با استفاده از تابع GaussianBlur در کتابخانه OpenCV انجام داده شده است.

```
L = cv2.GaussianBlur(img,(0,0),sigma)
```

که sigma انحراف معیار این فیلتر می‌باشد. برای این منظور سیگما برابر با ۱۵ درنظر گرفته شد. که این مقدار با سعی و خطی و نسبت به محدوده اعدادی که به طور معمول استفاده می‌شوند مقدار دهی شد.

شکل ۳ روشنایی(luminance) به دست آمده از تصویر آزمایشی

با داشتن تخمین از روشنای تصویر حال می‌توانیم مقدار MSR را محاسبه کنیم. داریم:

$$R_{MSR_i}(x,y) = \sum_{n=1}^{K} \omega_k R_{SSR_i} = \sum_{n=1}^{K} \omega_k \{\log S_i(x,y) - \log[L_i(x,y)]\}$$

پیاده سازی این بخش به صورت زیر است:

```
# \omega is 1/3 and equal for all
    retinex = np.zeros_like(img)
    for s in sigmas:
        SSR , L = singleScale(img, s)
        retinex += SSR

    MSR = retinex/len(sigmas)
    return MSR
```

همانطور که در بخش کامنت این کد مشخص شده است، مقدار ω برای هر سه فیلتر یک سوم درنظر گرفته شده است.

حال برای محاسبه MSRCR خواهیم داشت:

$$R_{MSRCR_i}(x,y) = C_i(x,y) R_{MSR_i}(x,y)$$

که در این رابطه مقدار C با مراجعه به مرجع[1] به صورت زیر محاسبه می‌شود:

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

$$I'_i(x,y) = \frac{I_i(x,y)}{\Sigma_{j=1}^{S} I_j(x,y)}$$

$$C_i(x,y) = \beta \log[\alpha I'_i(x,y)]$$

که همانطور که در این مقاله ذکر شده است مقادیر آلفا و بتایی که بنظر می‌آید برای تمام صحنه ها عملکرد خوبی دارند توسط مولفان اصلی معرفی شده است. در این پیاده سازی از آلفا و بتا برابر با ۱۲۵ و ۴۶ استفاده شده است.

پیاده سازی این بخش به صورت زیر است:

```
    img_sum = np.sum(img,axis=2,keepdims=True)

    i_prime = img / img_sum

    color_restoration = beta * (np.log10(alpha*i_prime))
```

متاسفانه برای محاسبه $MSRCR$ طبق معادلات مطرح شده توسط مقاله مورد بررسی به نتایج قابل قبولی نرسیدیم که در نتیجه آن به پیاده سازی این قسمت از مقاله اصلی $MSRCR[1]$ پرداختیم که به صورت زیر است:

$$R_{MSRCR_i}(x,y) = C_i(x,y) R_{MSR_i}(x,y)$$

$$R_{MSRCR_i}(x,y) = 255 \frac{R_{MSRCR_i}(x,y) - \min_i(\min_{(x,y)} R_{MSRCR_i}(x,y))}{\max_i(\max_{(x,y)} R_{MSRCR_i}(x,y)) - \min_i(\min_{(x,y)} R_{MSRCR_i}(x,y))}$$

```
          for i in range(MSRCR.shape[2]):
        MSRCR[:, :, i] = (MSRCR[:, :, i] - np.min(MSRCR[:, :, i])) /
(np.max(MSRCR[:, :, i]) - np.min(MSRCR[:, :, i])) * 255

MSRCR = np.uint8(np.minimum(np.maximum(MSRCR, 0), 255)) # clipping
```

و پیاده سازی اصلی مقاله مورد بررسی به صورت زیر است:

$$R_{MSRCR_i}(x,y) = C_i(x,y) R_{MSR_i}(x,y)$$

$$Min = Mean(R_{MSR_i}) - Dynamic \,.\, Var(R_{MSR_i})$$

$$Max = Mean(R_{MSR_i}) + Dynamic \,.\, Var(R_{MSR_i})$$

$$R_{MSRCR_i}(x,y) = \sum_i 255 \,.\, \frac{R_{MSR_i} - Mean(R_{MSR_i}) + Dynamic \,.\, Var(R_{MSR_i})}{(2 \,.\, Dynamic \,.\, Var(R_{MSR_i})}$$

```
MSRCR = np.zeros_like(img)
Dynamic = 2
for i in range(MSR.shape[2]):
      MSRCR[:, :, i] = 255 * (   (MSR[:, :, i] - np.mean(MSR[:, :, i]) +
Dynamic*np.var(MSR[:, :, i]))   / (2*Dynamic*np.var(MSR[:, :, i]))    )
```

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

```
MSRCR = np.uint8(np.minimum(np.maximum(MSRCR, 0), 255))
```

خروجی MSRCR:

شکل ۴ خروجی MSRCR برای تصویر آزمایشی

BF:

پیاده سازی فیلتر Bilateral با استفاده از تابع فراهم شده در کتابخانه OpenCV انجام شده است. ابرپارامتر های مطرح شده در مقاله نیز در این بخش گنجانده شده‌اند. داریم:

```
def bilateral_filter(img, filter_radius=15, sigma_d=5, sigma_r=0.03):
    sigma_color = sigma_r * 255  # Convert to the range of [0, 255]
    sigma_space = sigma_d
    return cv2.bilateralFilter(img, filter_radius, sigma_color,
sigma_space)
```

LP:

در این بخش به پیاده سازی هرم لاپلاسین که جزئیات تئوری آن در گزارش فنی به تفصیل ذکر شده است می‌پردازیم. همانطور که می‌دانیم برای این کار نیاز است که ابتدا هرم گوسی تصویر را تشکیل دهیم که این کار را می‌توان با استفاده از دو تابع cv2.pyrDown و cv2.pyrUp انجام داد. همانطور که از نام این توابع پیدا است تابع اول مرحله بعدی هرم گوسی را می‌سازد و تابع دوم برای تولید هرم لاپلاسین که نیاز به بزرگ کردن تصویر کوچک تر دارد می‌پردازد. کد این بخش به صورت زیر است:

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

```python
def create_gaussian_pyramid(MSRCR_img):
    gaussian = []
    gaussian_layer= MSRCR_img.copy()
    gaussian.append(gaussian_layer)
    for i in range(3):
        gaussian_layer = cv2.pyrDown(gaussian_layer)
        gaussian.append(gaussian_layer)
        # display_image_in_actual_size(gaussian_layer)
    return gaussian
```

شکل ۵ هرم گوسی

```python
def create_laplacian_pyramid(MSRCR_img):
    gaussian = create_gaussian_pyramid(MSRCR_img)
    laplacian = [gaussian[-1]]
    for i in range(3,0,-1):
      size = (gaussian[i - 1].shape[1], gaussian[i - 1].shape[0])
      gaussian_expanded = cv2.pyrUp(gaussian[i], dstsize=size)
      laplacian_layer = cv2.subtract(gaussian[i-1], gaussian_expanded)
      laplacian.append(laplacian_layer)
      # display_image_in_actual_size(laplacian_layer)
    return laplacian
```

شکل ۶ هرم لاپلاسین

شکل ۷ تصویر حاصل از BF و LP

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

GA:

این بخش اشاره به اصلاح گاما(gamma correction) دارد. این بخش را می‌توان با پیاده سازی معادلات زیر انجام داد:

$$sum = m.n.p$$

$$\gamma = \frac{\sum_{x=1}^{m}\sum_{y=1}^{n}\sum_{z=1}^{p} L(x,y,z)}{sum\ .N}$$

$$Ga_i = L'(x,y) = c[\ L(x,y)\ ]^{\gamma}$$

```
sum = img_L.shape[0]*img_L.shape[1]*img_L.shape[2]
N = 127
sum_of_values = np.sum(img_L)
gamma = sum_of_values / (sum * N)
c = 1
Ga = c * np.power(img_L, gamma)
# Normalize image to range [0, 1]
Ga = (Ga - np.min(Ga)) / (np.max(Ga) - np.min(Ga))
# back to uint8
Ga = (Ga * 255).astype(np.uint8)
```

شکل ۸ روشنایی تصویر(luminanc) پس از اعمال اصلاح گاما(gamma corecttion)

عملیات حسابی بین خروجی های مختلف و جمع وزن دار نهایی:

```
LL = img_as_float64(bf) + Lambda*img_as_float64(laplacian_pyramid[-1])

S = img_as_float64(Ga) * img_as_float64(LL)

alpha = 0.7

result = cv2.addWeighted(img_as_float64(LL),alpha,img_as_float64(S),1-
alpha,0)
```

نتایج و بررسی:

شکل ۹ تصاویر اولیه(همراه با مه) سمت راست و تصاویر مه‌زدایی شده به دست آمده(سمت چپ)

همانطور که از نتایج بالا مشخص است توانسته ایم تا حد بسیار خوبی الگوریتم مقاله را پیاده سازی کنیم. لازم به ذمر می‌باشد که از آنجایی که برای تمام تصاویر از ابرپارامتر های ثابتی استفاده کرده ایم در برخی از موارد به بهترین حالت ممکن دست پیدا نکرده‌ایم اما با همین فرض ساده نیز توانسته ایم نتایج بسیار خوبی از منظر دید انسان(subjective) به دست آوریم. حال به سراغ معیار های معرفی شده در مقاله یعنی AG و IE می‌رویم.

معیار میانگین گرادیان نرخ تغییر چگالی تصویر در جهت های مختلف را می‌دهد که در واقع نشان دهنده وضوح تصویر است. در نتیجه زمانی که AG بالاتر باشد نشان دهنده‌ی وضوح بیشتر آن تصویر است.

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

معیار اطلاعات انتروپی، میانگین اطلاعات تصویر است که نشان دهنده چگونگی توزیع و تجمع ویژگی های تصویر است. هرچه مقدار IE بیشتر باشد نشان دهنده غنی تر بودن اطلاعات رنگی است.

پیاده سازی این معیار ها به صورت زیر است:

$$AG = \frac{1}{(M-1)(N-1)} \sum_{i=1}^{M-1} \times \sum_{j=1}^{N-1} \sqrt{\frac{(f(i+1,j)-f(i,j))^2+(f(i,j+1)-f(i,j))^2)}{2}}$$

که پیاده سازی آن به شکل زیر بر اساس معادله بالا انجام شد:

```
def AG(image):

  M = image.shape[0]
  N = image.shape[1]

  total = 0
  for i in range( 1,M-1):
    for j in range(1,N-1):
      tmp1 = (image[i+1,j] - image[i,j])**2
      tmp2 = (image[i,j+1] - image[i,j])**2

      tmp3 = tmp1 + tmp2

      total += math.sqrt(tmp3 / 2)

  return total/((M-1) * (N-1))
```

و برای آنتروپی اطلاعات داریم:

$$IE = -\Sigma_{i=0}^{n} p_i \log_2 p_i$$

پیاده سازی آن با استفاده از هیستوگرام(احتمال پیکسل ها) و تابع entropy در کتابخانه scipy.stats انجام شده است.

```
from scipy.stats import entropy
def IE(image):
    # histogram
    hist, _ = np.histogram(image, bins=256, range=(0, 256), density=True)

    # entropy
    ie = entropy(hist, base=2)
    return ie
```

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

نتایج به شرح زیر می‌باشند:

AG:

```
/content/Capture.PNG
0.012408455263756809
0.011676221103940928

/content/Capture1.PNG
0.025685671232169804
0.025716674579556762

/content/Capture3.PNG
0.008346229330361039
0.006649593796185936

/content/Capture4.PNG
0.01055087698600658
0.007839971018674114

/content/Capture5.PNG
0.025171415730522558
0.028524943298824195

/content/Capture6.PNG
0.005734919761413598
0.004042396408570458

/content/Capture7.PNG
0.023946602579916803
0.018980168067409382

/content/Capture8.PNG
0.012646007497042219
0.0140432774980918

/content/Capture9.PNG
0.010730465981336932
0.022965786407847764
```

IE:

```
/content/Capture.PNG
0.022875979046570613
0.0007329049606171476

/content/Capture1.PNG
0.09532640180193461
0.00013981007619679253

/content/Capture3.PNG
0.040859101397703074
0.0002559121812790193

/content/Capture4.PNG
```

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰

```
0.03415199535105298
0.00011927383326524555

/content/Capture5.PNG
0.00017447002580860975
0.00012007121888201253

/content/Capture6.PNG
0.01876140283571865
6.262009070416071e-05

/content/Capture7.PNG
0.018702182911972835
6.293076496724075e-05

/content/Capture8.PNG
0.00037961184976006415
0.00012007121888201253

/content/Capture9.PNG
0.001372681810985882
6.282586217947822e-05
```

عدد اول مربوط به تصویر همراه با مه و عدد دوم مربوط به تصویر مه‌زدایی شده می‌باشد.

متاسفانه در تعداد از موارد معیار ها نشان دهنده آن هستند که تصاویر مه‌زدایی شده بهبودی از نظر معیار نداده اند که این موضوع می‌تواند به دلیل نحوه جمع آوری تصاویر(screenshot) و یا تنظیم نبودن هایپرپارامتر ها و یا نقص در نحوه محاسبه معیار ها باشد.

لازم به ذکر می‌باشد که روند کلی مقاله به خوبی پیاده سازی شده است و نتایج مطلوبی از نظر دید انسان به دست آمده است و این مشکلات نیز قطعا با ویرایش های جزئی قابل حل می‌باشند.

منابع:

[1] A. Petro, C. Sbert, J. Morel. "Multiscale Retinex," in *Image Processing On Line*, pp. 71–88, 2014.

https://www.projectpro.io/recipes/do-laplacian-pyramids-work-opencv

علیرضا دلاوری
۴۰۲۱۳۱۰۲۰