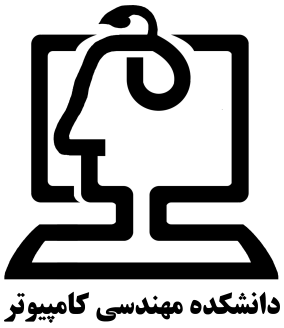
به نام خدا



دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر

استاد درس: دکتر صفابخش

پاییز ۱۴۰۰

درس بینایی ماشین

سروش مهدی
شماره دانشجویی: ۹۹۱۳۱۰۵۰

گزارش تمرین اول

فهرست مطالب

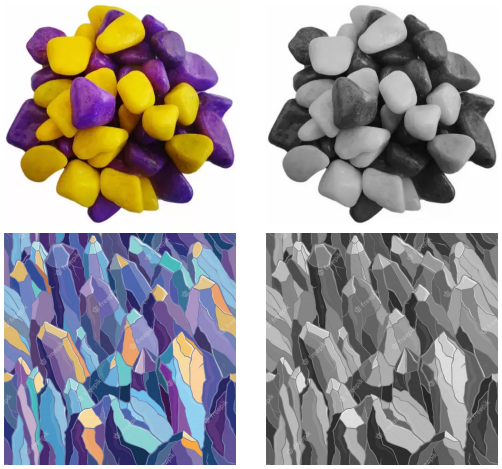
۱ سوال اول

برای تبدیل تصاویر رنگی یه تصاویر سطوح خاکستری، ابتدا آنها را بازخوانی کرده سپس با تبدیل فضای رنگی این تبدیل را انجام می دهیم.

```python
#reading images
hist1 = cv2.imread('Hist1.webp')
hist2 = cv2.imread('Hist2.webp')
#converting to grayscale
hist1_gray = cv2.cvtColor(hist1, cv2.COLOR_BGR2GRAY)
hist2_gray = cv2.cvtColor(hist2, cv2.COLOR_BGR2GRAY)
#showing images
for img in [hist1, hist1_gray, hist2, hist2_gray]:
    cv2.imshow('image',img)
    cv2.waitKey(0)
cv2.destroyAllWindows()
#saving grayscale images
cv2.imwrite('hist1_gray.jpg' , hist1_gray)
cv2.imwrite('hist2_gray.jpg', hist2_gray)
```

Listing :۱

ابتدا خواندن تصاویر از فایل ورودی توسط تابع imread انجام می شود. سپس با فراخوانی تابع cvtColor با آرگومان BGR2GRAY تبدیل فضای رنگی انجام می شود. بعد در یک حلقه تصاویر رنگی و سطح خاکستری نمایش داده می‌شوند و در نهایت تصاویر ذخیره می‌گردند. در این تمرین توجه به این نکته ضروری است. که فضای رنگی پیشفرض در کتابخانه OpenCV به صورت BGR می باشد، نه RGB .



شکل ۱: تصاویر رنگی به همراه تصاویر سطح خاکستری انها

۲ سوال دوم

در این بخش دو روش پیاده‌سازی شده که در ادامه هر دو انها را توضیح می‌دهیم. ابتدا پس زمینه تصویر را برای استفاده در هر دو روش از سفید به مشکی تغییر میدهیم زیرا در ادامه کارمان راحت تر میشود. کد این بخش به صورت زیر میباشد.

```
hist1 = cv2.imread('Hist1.webp')
#creating a mask for white color using trasholding
#in (200 , 255) range on colors
mask = cv2.inRange(hist1, np.array([200, 200, 200], dtype="uint8"), np.array([255, 255, 255], dtype="uint8"))

mask2 = np.zeros_like(hist1)
mask2[:, :, 0] = mask
mask2[:, :, 1] = mask
mask2[:, :, 2] = mask
# hist and (not mask) -> turns white color to black
hist1 = cv2.bitwise_and(hist1, 255 - mask2)
```

Listing :۲

در این کد ابتدا با تعیین محدوده رنگ سفید بین ۲۰۰ و ۲۵۵ یک ماسک برای رنگ سفید ایجاد میکنیم و سپس روی تصویر اولیه و معکوس این ماسک یک اپراتور and پیکسل به پیکسل اجرا میکنیم. نتیجه این بخش به صورت زیر هست.

شکل ۲: تصویر اولیه به همراه تصویری که پس زمینه ان مشکی شده

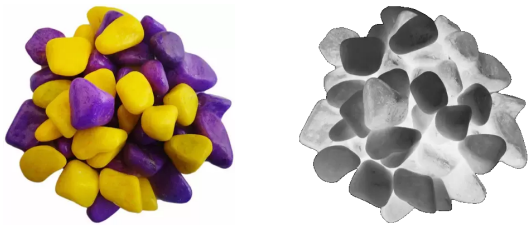
کد روش اول به صورت زیر میباشد:

```
#grayscale transform
hist1_gray = cv2.cvtColor(hist1, cv2.COLOR_BGR2GRAY)
#inverting grayscale photo
hist1_gray_inverted = cv2.bitwise_not(hist1_gray)
```

Listing :۳

در سوال اول مشاهده کردیم با تبدیل تصویر اصلی به تصویر سطح خاکستری رنگ های زرد به سمت رنگ سفید و رنگ های بنفش به سمت رنگ مشکی میروند. پس اگر این تصویر سطح خاکستری را معکوس کنیم

انتظار داریم رنگ های زرد به سمت مشکی و رنگ های بنفش به سمت سفید بروند. نتیجه این بخش به صورت زیر میباشد.



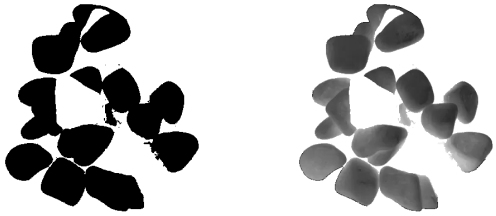
شکل ۳: تصویر اولیه به همراه تصویر حاصل از روش اول

کد روش دوم به صورت زیر میباشد:

```
#converting color space to HSV
hist1_hsv = cv2.cvtColor(hist1 , cv2.COLOR_BGR2HSV)
#creating a mask for yellow color
mask3 = cv2.inRange(hist1_hsv, (15,0,0), (36, 255, 255))
# not (hist1gray and yellow mask) -> first turns yellow color
# to white then invert the photo
hist1_masked = cv2.bitwise_not(cv2.bitwise_and(hist1_gray , mask3))
#inverting yellow mask
mask3_inv = cv2.bitwise_not(mask3)
```

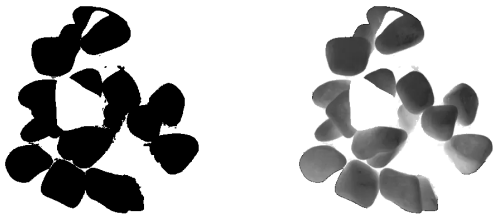
Listing :۴

در این روش ابتدا یک ماسک برای رنگ زرد میسازیم. برای اینکار تصویر را به فضای رنگی hsv میبریم زیرا جداسازی رنگ ها در ان ساده تر میباشد. سپس یک ماسک برای رنگ زرد میسازیم که مقدار متغیر hue در ان بین ۱۵ و ۳۶ تغییر میکند. در این بخش دو تصویر بدست امده که اولی معکوس تصویر حاصل از اپراتور and بین تصویر خاکستری و ماسک رنگ زرد میباشد. دومی نیز معکوس ماسک رنگ زرد میباشد.این تصاویر بصورت زیر هستند



شکل ۴: تصاویر بدست امده از روش دوم

اگر مقدار hue از ۱۰ تا ۳۵ تغییر کند تصاویر به صورت زیر خواهند شد که در این حالت سنگ زرد وسط تصویر که زیر بقیه سنگ هاست بهتر مشخص شده.



شکل ۵: تصاویر بدست امده از روش دوم

۳ سوال سوم

یکی از مشکلات استفاده از فضای رنگی RGB و برخی دیگر از فضاهای رنگی مشابه، عدم وجود کانال رنگی مجزا برای روشنایی در تصویر است. در این تمرین برای کار با روشنایی تصویر باید ان را به فضای رنگی دیگر منتقل کرد که روشنایی را به صورت کانالی جدا کدگذاری میکند. برای اینکار ما از دو فضای رنگی l*a*b* و YCBCR استفاده میکنیم. در هر دوی این فضاها کانال اول به روشنایی تعلق دارد.

بعد از تغییر فضای رنگی تصاویر روی کانال مربوط به روشنایی ان ها عملیات متعادل سازی هیستوگرام را انجام میدهیم سپس تصویر را به فضای رنگی اولیه میبریم.

برای این بخش از تابع زیر استفاده میکنیم.

```python
def HE(img , color_space , inv_color_space , name ):

    # convert from RGB to color_space
    img = cv2.cvtColor(img,color_space)

    # equalize the histogram of the luminance channel
    img[:, :, 0] = cv2.equalizeHist(img[:, :, 0])

    # convert back to RGB
    equalized_img = cv2.cvtColor(img, inv_color_space)

    cv2.imshow('equalized_img', equalized_img)
    cv2.imshow('equalized_luminance_channel', equalized_img[:,:,0])
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite(name + '.png' , equalized_img)
```

Listing :۵

نتایج این بخش به صورت زیر هستند:

شکل ۶: تصاویر اولیه

شکل ۷: تصاویرنتایج با استفاده از فضای LAB

شکل ۸: تصاویر نتایج با استفاده از فضای ycbcr

اگر در هیستوگرام پراکندگی میزان روشنایی بالا باشد عملیات متعادل سازی هیستوگرام خوب جواب نمیدهد و باید از متعادل سازی سازگار هیستوگرام استفاده کنیم. برای این منظوراز تابع صفحه بعد استفاده میکنیم.

نتایج این بخش در صفحه بعد امده اند.

مشاهده میشود در تصویری که با نور مصنوعی گرفته شده در مورد گوشه بالا سمت راست نصویر این روش عملکرد بهتری نسبت به روش قبلی دارد.

```python
def AHE(img , color_space , inv_color_space , name):

    # convert from RGB to color_space
    img = cv2.cvtColor(img,colorSpace)


    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

    # equalize the histogram of the luminance channel
    img[:, :, 0] = clahe.apply(img[:, :, 0])

    # convert back to RGB
    equalized_img = cv2.cvtColor(ycrcb_img,inv_color_space)

    cv2.imshow('equalized_img', equalized_img)
    cv2.imshow('equalized_luminance_channel', equalized_img[:,:,0])
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite(name + '.png', equalized_img)
```

Listing :۶

شکل ۹: تصاویرنتایج با استفاده از فضای LAB

شکل ۱۰: تصاویر نتایج با استفاده از فضای ycbcr

۴ سوال چهارم

کد این بخش به صورت زیر می‌باشد. در این کد ابتدا یک شی از VideoCaptur ایجاد می‌کنیم سپس مقدار n را مشخص می‌کنیم و سپس یک لیست برای نگهداری فریم ها ایجاد می‌کنیم. بعد در یک حلقه بی‌نهایت فریم ها را از وب‌کم میخوانیم و اگر فریم به صورت درست خوانده شده باشد پیکسل های ان را به n تقسیم می‌کنیم که دلیل اینکار برای میانگین گیری در مرحله بعد میباشد و این ماتریس را به انتهای لیست اضافه می‌کنیم. سپس چک میکنیم اگر تعداد فریم های خوانده شده تا این لحظه از n بزرگتر باشد روی n فریم اخر میانگین گیری میکنیم. در اینجا چون قبا فریم ها را به n تقسیم کردیم جمع مقادیر جدید برابر میانگین فریم ها میباشد. سپس نوع متغیر های ماتریس میانگین را به uint8 تغییر میدهیم زیرا فریم خوانده شده از وبکم پیکسل هایش با این فرمت ذخیره شده‌اند.

```
cap = cv2.VideoCapture(0)
n = 15 #num of frames for averaging
images = [] #list to save frames

while(True):
    ret, frame = cap.read()

    if ret: #if frame has fecthed correctly add it to list of frames
        images.append(frame/n)

    if len(images) > n: #if number of fecthed frames is bigger than n do averaging
        result = np.sum(images[-n:], axis = 0).astype(np.uint8)
        #showing result
        cv2.imshow('result', result)
    #showing main frames
    cv2.imshow('video', frame)
    if cv2.waitKey(30) == 27: #waiting 30 miliseconds
        # ESC pressed
        print("Escape hit, closing...")
        break
#releasing cam
cap.release()
cv2.destroyAllWindows()
```

Listing :۷

بعد هر دو تصویر اصلی و میانگین گیری شده را نمایش می‌دهیم و ۳۰ میلی ثانیه صبر میکنیم برای تکرار این روند. همچنین دکمه ESC نیز برای خروج از حلقه در نظر گرفته شده.

با افزایش پارامتر n مشاهده میشود که حرکات smooth تر شده وهر چقد این پارامتر را افزایش دهیم حرکت یک شی بیشتر زمان نیاز دارد برای جا به جا شدن کامل شی زیرا فریم های بیشتری را از گذشته دورتر در میانگین گیری دخالت میدهیم. اگر بخواهیم حرکت یک شی را در یک تصویر مشخص کنیم با استفاده از این روش میتوانیم با افزایش n گذشته دورتری از حرکت را در اختیار داشته باشیم.

۵ سوال پنجم

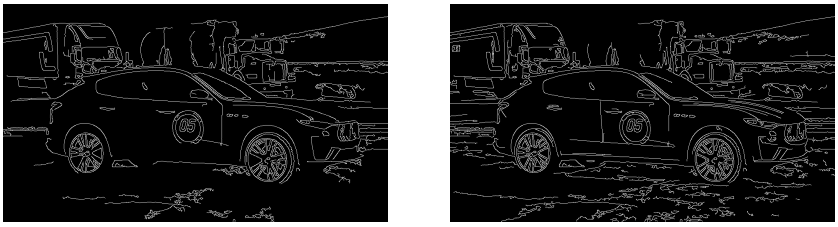
کد این بخش به صورت زیر میباشد:

```
import cv2
import numpy as np


edge = cv2.imread('edge.webp')
edge_avg = cv2.blur(edge, (9,9))
edge_gaussian = cv2.GaussianBlur(edge, (9,9), cv2.BORDER_DEFAULT)
edge_median = cv2.medianBlur(edge, 9)
edge_bilateral = cv2.bilateralFilter(edge, 9, 5000, 5000)
edge_pyr = cv2.pyrUp(cv2.pyrDown(edge))

cv2.imwrite('canny_pyr.png' ,cv2.Canny(edge_pyr, 100, 300))
cv2.imwrite('canny_pyr2.png' ,cv2.Canny(edge_pyr, 50, 400))
cv2.imwrite( 'canny1.png' ,cv2.Canny(edge, 50, 150))
cv2.imwrite('canny2.png' ,cv2.Canny(edge, 500, 1000))
cv2.imwrite('canny_avg.png' , cv2.Canny(edge_avg, 50, 150))
cv2.imwrite('canny_gaussian.png' ,cv2.Canny(edge_gaussian, 50, 150))
cv2.imwrite('canny2_med.png' ,cv2.Canny(edge_median, 50, 150))
cv2.imwrite('canny2_bi.png' ,cv2.Canny(edge_bilateral, 50, 150))
```

Listing :۸

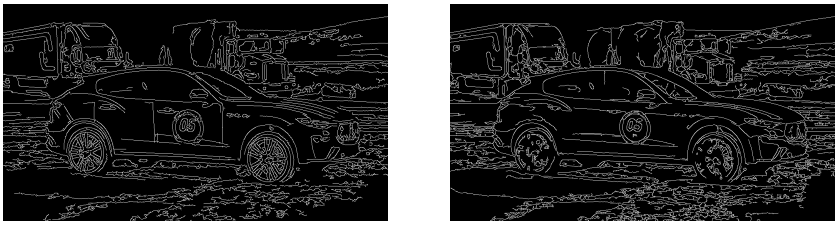
در پردازش لبه ها، نویز اثر بسیار مخربی دارد. در اینجا نیز میتوان گفت بافت اسفالت باعث نوعی نویز شده که کار تشخیص لبه را سخت میکند. برای رفع این مشکل از فیلتر های رفع نویز مختلف و روش تغییر رزولوشن هرمی استفاده میکنیم که جزییات تصویر را کمتر میکند و فقط مولفه های مهم را مشخص میکند. همچنین از افزایش threshold در فیلتر تشخیص لبه (Canny) نیز استفاده میکنیم.
نتایج این بخش به صورت زیر هستند:

شکل ۱۱: نتایج لبه یاب کنی با سطح استانه ۵۰-۱۵۰ در راست و ۵۰۰-۱۰۰۰ در چپ



شکل ۱۲: نتایج کنی به همراه روش تغییر رزولوشن با سطح استانه ۱۰۰-۳۰۰ در راست و ۵۰-۴۰۰ در چپ

شکل ۱۳: نتایج کنی با روش رفع نویز میانگین گیری در راست و گاوسی در چپ



شکل ۱۴: نتایج کنی با روش رفع نویز میانه گیری در راست و bilateral در چپ

مشاهده میشود که روش های رفع نویز گاوسی و میانگین گیری و همچنین روش کاهش رزولوشن عملکرد بهتری نسبت به بقیه روش ها دارند.