به نام خالق رنگین کمان

ستاره باباجانی - 9521109

سوال 1: هنگامی که یک تصویر در مقیاس خاکستری را به 64 سطح کمی می کنیم، محدوده مقادیر روشنایی پیکسل را از 0 تا 255 به محدوده کوچکتر کاهش می دهیم. .برای یافتن محدوده جدید، محدوده اصلی را باید بر تعداد سطوح تقسیم بکنیم.

محدوده اصلی: 0 تا 255

تعداد سطوح پس از کوانتیزه شدن: 64

$$\text{محدوده جدید} = \frac{\text{محدوده اصلی}}{\text{تعداد سطوح}}$$

$$\text{محدوده جدید} = \frac{255}{64}$$

حاصل این کسر برابر با 3.984 میباشد، اما از آنجایی که مقادیر پیکسل باید اعداد صحیح باشند، آن را به 4 گرد می کنیم. بنابراین، هر بازه در محدوده جدید تقریباً 4 واحد خواهد بود. محدوده جدید از 0 شروع می‌شود و برای هر سطح، 4 تا افزایش می‌یابد که به ما محدوده‌ای از [0، 4، 8، 12، ...، 252] می‌دهد. بنابراین، محدوده جدید مقادیر روشنایی پیکسل پس از کمیت کردن به 64 سطح [0 تا 252] است.

در مورد کیفیت تصویر، کوانتیزاسیون تعداد سطوح متمایز در تصویر را کاهش می دهد. در این حالت با کاهش تصویر به 64 سطح، عملاً میزان جزئیات تصویر را کاهش می دهیم. این کاهش جزئیات می تواند منجر به از دست دادن کیفیت تصویر شود، به خصوص اگر تصویر اصلی دارای تغییرات جزئی در روشنایی باشد که اکنون با سطوح کمتری تقریب می شود. میزان تأثیر بر کیفیت تصویر به عواملی مانند محتوای اصلی

تصویر و الزامات خاص برنامه بستگی دارد. به طور کلی، سطوح بالاتر کوانتیزاسیون منجر به کاهش قابل توجهی در کیفیت تصویر می شود.

سوال 2: برای ثبت تصویری واضح از یک پرنده در حال پرواز، عکاس باید از سرعت شاتر مناسب استفاده کند. سرعت شاتر تعیین می کند که حسگر دوربین در هنگام ثبت تصویر چه مدت در معرض نور قرار می گیرد. انتخاب سرعت شاتر به عوامل مختلفی از جمله سرعت حرکت پرنده، میزان تاری حرکتی مورد نظر و شرایط نور بستگی دارد.

برای گرفتن تصویر واضح از پرنده در حال پرواز، باید از سرعت شاتر سریع استفاده کنیم. این به منجمد کردن حرکت پرنده کمک می کند و جزئیات واضح را بدون تاری حرکت تضمین می کند. سرعت شاتر دقیق مورد نیاز به سرعت حرکت پرنده بستگی دارد. به عنوان یک دستورالعمل کلی:

1. برای پرندگانی که کندتر حرکت می کنند یا پرندگانی که با سرعت متوسط پرواز می کنند، سرعت شاتر در حدود 500/ 1 تا 1000 / 1 ثانیه ممکن است کافی باشد.
2. برای پرندگانی که سریع‌تر حرکت می‌کنند یا پرندگانی که مانورهای سریع انجام می‌دهند، سرعت شاتر 1000 / 1 ثانیه یا سریع‌تر ممکن است برای منجمد کردن مؤثر حرکت ضروری باشد.

استفاده از سرعت شاتر سریع باعث می شود که حرکت پرنده به وضوح و بدون تاری ثبت شود و در نتیجه تصویری واضح و واضح به دست می آید. با این حال، تعادل

سرعت شاتر با سایر تنظیمات نوردهی مانند دیافراگم و ISO برای رسیدن به نوردهی مناسب و حفظ کیفیت تصویر ضروری است.

در حقیقت یک trade off وجود دارد:

1. نور: سرعت شاتر بیشتر نور کمتری را وارد می کند. در شرایط نور کم، عکاس ممکن است نیاز به تنظیم ISO (حساسیت حسگر) برای جبران داشته باشد.ISO بالاتر می تواند دانه بندی تصویر را معرفی کند، بنابراین یافتن تعادل مناسب بسیار مهم است.
2. دیافراگم: گزینه دیگر باز کردن دیافراگم (عدد f پایین تر) برای اجازه دادن به نور بیشتر است. این کار عمق میدان کمتری ایجاد می کند، بنابراین ممکن است برخی از قسمت های تصویر خارج از فوکوس باشند. عکاس باید تصمیم بگیرد که آیا پس زمینه کمی تار برای یک پرنده کاملاً تیز قابل قبول است یا خیر.

به طور خلاصه، برای ثبت یک تصویر واضح از یک پرنده در حال پرواز، از سرعت شاتر سریع مناسب با سرعت حرکت پرنده استفاده میکنیم تا عمل ثابت شود و از تاری حرکت جلوگیری شود.

منابع کمکی:

Chat.openai & gemini

سوال 3: برای بهبود کیفیت عکس گرفتن از تعقیب و گریز پر سرعت یوزپلنگ و در عین حال دستیابی به حس سرعت و دید گسترده، میتوان رویکرد زیر را در نظر گرفت:

1. افزایش سرعت شاتر: از سرعت شاتر سریعتر برای منجمد کردن حرکت یوزپلنگ و ایجاد یک تصویر واضح استفاده کنیم. از آنجایی که یوزپلنگ به سرعت در حال حرکت است، سرعت شاتر حداقل 1000 / 1 تا 2000 / 1 ثانیه یا حتی سریعتر برای تصویربرداری بدون تاری مناسب است.
2. تنظیم عمق میدان: برای دستیابی به پس زمینه تار و تمرکز روی یوزپلنگ، از عمق میدان کم استفاده میکنیم. این را می توان با استفاده از دیافراگم باز (عدد f-stop کوچک) روی دوربین به دست آورد. برای مثال، استفاده از دیافراگم f/2.8 یا بیشتر، به محو شدن پس‌زمینه کمک می‌کند و در عین حال یوزپلنگ را در فوکوس واضح نگه می‌دارد.
3. استفاده از فوکوس خودکار پیوسته: حالت فوکوس خودکار مداوم دوربین را فعال کنیم تا سوژه متحرک (یوزپلنگ) در طول تعقیب و گریز در فوکوس بماند. این ویژگی به دوربین این امکان را می دهد که به طور مداوم فوکوس را در حین حرکت سوژه تنظیم کند و اطمینان حاصل کند که یوزپلنگ حتی در صورت حرکت سریع، واضح باقی می ماند.
4. میدان دید مناسب را انتخاب کنیم: از فاصله کانونی بیشتر(بازتر) یا کوچکنمایی استفاده کنیم تا نمای وسیع تری از مناظر را به تصویر بکشیم در حالی که همچنان یوزپلنگ را در کادر برجسته نگه داشته شود. این به عکاس این امکان را می دهد که در عین تأکید بر عمل تعقیب و گریز، محیط بیشتری را در بر بگیرد.
5. تکنیک panning را تمرین کنیم: هنگام عکاسی از سوژه هایی که سریع حرکت می کنند مانند یوزپلنگ در حال حرکت، از این تکنینک استفاده می کنیم. این شامل ردیابی سوژه با دوربین در حالی که از سرعت شاتر کندتر استفاده می شود، است. با دنبال کردن حرکت یوزپلنگ با دوربین در حالی که

شاتر را فشار می دهیم، پس زمینه تار می شود در حالی که سوژه نسبتا واضح باقی می ماند و حس سرعت و حرکت در تصویر ایجاد می شود.

6. از burst mode استفاده کنیم: از حالت پشت سر هم دوربین برای گرفتن مجموعه ای از عکس های شلیک سریع در حین تعقیب و گریز استفاده کنیم. این کار شانس گرفتن عکسی کاملاً زمان‌دار و متمرکز از یوزپلنگ را افزایش می‌دهد.

سوال 5: الف) همان طور که در فایل مربوطه ذکر شد، بین نشان دادن عکس با استفاده از opencv و matplotlib تفاوت هایی وجود دارد که به شرح زیر است:

- OpenCV: OpenCV یک کتابخانه بینایی کامپیوتری است که بر پردازش و تجزیه و تحلیل تصویر متمرکز شده است. cv2.imshow() یک تابع ارائه شده توسط OpenCV برای نمایش تصاویر است. به طور پیش‌فرض، OpenCV از ترتیب رنگ BGR (آبی-سبز-قرمز) برای نمایش تصاویر استفاده می‌کند که با ترتیب رنگ رایج RGB (قرمز-سبز-آبی) متفاوت است.
- Matplotlib: Matplotlib یک کتابخانه ترسیم جامع برای پایتون است که به طور گسترده برای ایجاد تجسم و نمودار استفاده می شود. Matplotlib عملکردهایی را برای نمایش تصاویر از طریق تابع imshow()خود که بخشی از ماژول matplotlib.pyplot است فراهم می کند. Matplotlib از ترتیب رنگ RGB برای نمایش تصاویر استفاده می کند، که قرارداد استاندارد است.

پس طبق چیزهایی که گفته شد، باتوجه به ترتیب رنگ متفاوت بین آنها، ممکن است رنگ‌ها در مقایسه با نمایش همان تصویر با استفاده از Matplotlib عوض شده باشند.

همچنین `cv2.imshow()` یک رابط پنجره ساده برای نمایش تصاویر ارائه می دهد، در حالی که Matplotlib می تواند بسته به محیط، تصاویر را در یک نوت بوک Jupyter یا در یک پنجره جداگانه نمایش دهد. همچنین Opencv ممکن است عملکرد بهتری برای نمایش سریع تصاویر داشته باشد، به خصوص زمانی که با برنامه های بلادرنگ یا حجم زیادی از تصاویر سروکار داریم.

حال اگر بخواهیم تصویر matplotlib شبیه opencv شود، باید به ترتیب رنگ GBR تبدیل کنیم:

```
image_bgr = cv2.imread('ComputerDepartment.jpg')

# Convert BGR image to RGB (for displaying with Matplotlib)
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)

# Display the image using Matplotlib
plt.imshow(image_rgb)
plt.axis('off')  # Turn off axis labels
plt.show()
```

ب)

1. تصویر باید بصورت عمودی به معکوس عمودی اش متصل شود: ابتدا چک میکنیم تصویر اصلی وجود داشته باشد، سپس از دستور flip برای معکوس کردن استفاده میکنیم که در جهت ایندکس 0 آن است پس معکوس عمودی میشود. سپس با دستور vstack دو عکس را بهم میچسبانیم و نمایش میدهیم:

```
if I is not None:
    # Flip the main image vertically
    vertical_invert = cv2.flip(I, 0)

    # Stack the main image and its vertical invert vertically
    result_image = np.vstack((I, vertical_invert))

    # Display the result image
    cv2_imshow(result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Error: Unable to load the main image.")
```

2. معکوس عمودی تصویر اصلی به صورت عمودی به تصویر اصلی متصل شود: در این بخش همه چیز شبیه بخش قبلی هست و فقط در مرحله وصل کردن دو تصویر تغییرات داریم(تصویر اصلی را به ادامه تصویر معکوس شده میچسبانیم):

```
if I is not None:
    # Flip the original image vertically to create the vertical invert
    vertical_invert = cv2.flip(I, 0)

    # Stack the vertical invert and original image vertically
    result_image = np.vstack((vertical_invert, I))

    # Display the result image
    cv2_imshow(result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Error: Unable to load the original image.")
```

3. تصویر اصلی به صورت افقی به معکوس افقی اش متصل شود: در این بخش ابتدا با دستور flip و ایندکس 1 تصویر را بصورت افقی معکوس کرده و سپس تصویر معکوس شده را در کنار تصویر اصلی با کمک hstack قرار میدهیم:

```
if I is not None:
    # Flip the main image horizontally
    horizontal_invert = cv2.flip(I, 1)

    # Concatenate the main image and its horizontal invert horizontally
    result_image = np.hstack((I, horizontal_invert))

    # Display the result image
    cv2_imshow(result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Error: Unable to load the main image.")
```

4. معکوس افقی تصویر اصلی به صورت افقی به تصویر اصلی متصل شود: این قسمت مانند قسمت قبل است با این تفاوت که وقتی از hstack استفاده میکنیم، تصویر اصلی را به تصویر معکوس شده متصل میکنیم:

```
if I is not None:
    # Flip the main image horizontally
    horizontal_invert = cv2.flip(I, 1)

    # Concatenate the main image and its horizontal invert horizontally
    result_image = np.hstack((horizontal_invert, I))

    # Display the result image
    cv2_imshow(result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Error: Unable to load the main image.")
```

5. تغییر رنگ یکی از پنجره های ساختمان: در این بخش، ابتدا یک کپی از عکس اصلی گرفته شد و سپس مختصات x, y و طول و عمق پنجره را به دلخواه تنظیم شد. سپس طبق مختصات پیدا شده، رنگ آن را زرد کردم و نمایش دادم. در نهایت با دستور immwrite عکس جدید را در Colored_Window ذخیره کردم.

```
I = cv2.imread('ComputerDepartment.jpg', cv2.IMREAD_UNCHANGED)
copy_I = I.copy()

# Define the coordinates of the window
x, y, width, height = 271, 305, 30, 55

# Change the color of the window to yellow
copy_I[y:y+height, x:x+width] = [30, 255, 255]

# Display the image with the changed window color
cv2_imshow(copy_I)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

6. لوگو و نمایش کانال های رنگی: در این سوال ابتدا لوگوی اصلی را خوانده و نمایش دادیم:

```
image = cv2.imread('logo.jpg', cv2.IMREAD_UNCHANGED)

# Display the original image
cv2_imshow(image)
cv2.waitKey(5000)
cv2.destroyAllWindows()
```

سپس با دستور split موجود در کتابخانه opencv کانال های رنگی آن را جدا کرده و سپس برای اینکه هر کانال رنگی را نشان دهیم، مقادیر دو کانال رنگی دیگر را با استفاده از متغیر black_image صفر کردیم(یعنی برای کانال قرمز، کانال سبز و آبی را صفر کردیم):

```
# Split the image into its color channels
blue_channel, green_channel, red_channel = cv2.split(image)

# Create black images
black_image = np.zeros_like(blue_channel)
```

سپس برای نمایش هر کانال، مقدار آن کانال رنگی را با مقدار black_image برای دو کانال دیگرش مرج کردیم و در نهایت نشان دادیم:

```
# blue channel
blue_img = cv2.merge((blue_channel, black_image, black_image))

# Display the blue channel
cv2_imshow(blue_img)
cv2.waitKey(5000)
cv2.destroyAllWindows()
```

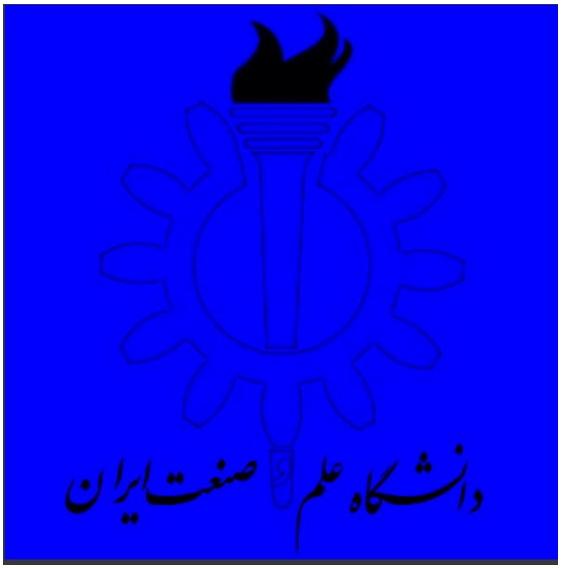
```
# green channel
green_img = cv2.merge((black_image, green_channel, black_image))

# Display the green channel
cv2_imshow(green_img)
cv2.waitKey(5000)
cv2.destroyAllWindows()
```

```
# red channel
red_img = cv2.merge((black_image, black_image, red_channel))

# Display the red channel
cv2_imshow(red_img)
cv2.waitKey(5000)
cv2.destroyAllWindows()
```

همان طور که در تصاویر فایل کد نمایش داده شده است، برای کانال آبی، هر رنگ آبی ای که در لوگوی اصلی وجود داشت، باقی ماند و مابقی رنگ ها به دلیل متغیر تعریف شده، سیاه شد. همچنین کادر دور عکس اصلی نیز آبی شد:



همچنین برای کانال سبز، چون رنگ سبزی در عکس اصلی موجود نبود، تنها کادر دور لوگو سبز شد و بقیه رنگ های لوگو مشکی شدند:

در نهایت برای کانال قرمز، مثل کانال آبی، کادر دور لوگو علامت آتش لوگو قرمز و بقیه لوگو مشکی شد: