تمرین سری اول بینایی کامپیوتر
ملیکا محمدی فخار - ۹۹۵۲۲۰۸۶

.۱
هنگامی که سطوح روشنایی یک تصویر را از ۲۵۶ سطح (۰ تا ۲۵۵) به ۶۴ سطح کاهش دهیم، تعداد مقادیر روشنایی ممکن برای هر پیکسل کاهش می‌یابد. از آنجایی که $256 \div 64 = 4$، هر بازه در تصویر کوانتیزه شده جدید، محدوده ۴ مقدار روشنایی از تصویر اصلی را پوشش می‌دهد.
درمورد کیفیت تصویر، کوانتیزه کردن مقدار اطلاعات مورد نیاز برای نمایش تصویر را کاهش می‌دهد. بنابراین، کیفیت تصویر نیز کاهش می‌یابد. زیرا هنگامی که تعداد سطوح روشنایی کاهش یابند، تصویر بخشی از جزئیات خود را از دست می‌دهد، به ویژه در مناطقی که تغییرات جزئی در روشنایی دارند. همچنین این عمل، کیفیت تصویر را در مناطقی که تغییرات تدریجی در شدت روشنایی دارند (مانند سایه‌ها) کاهش می‌دهد. برای مثال اگر مقادیر ۳ پیکسل کنار هم، به ترتیب ۰، ۱ و ۲ باشند، در تصویر کوانتیزه شده هر سه این مقادیر به ۰ تبدیل شده و تفاوت میان آن‌ها در تصویر از دست می‌رود.

.۲
پاسخ این سوال بستگی به سرعت حرکات پرنده دارد. به طور کلی از آن‌جا که پرندگان می‌توانند به سرعت بال‌های خود را هنگام پرواز به حرکت درآورند، برای جلوگیری از تار شدن عکس نیاز است تا سرعت شاتر به اندازه‌ای بالا باشد (مثلا ۰.۰۰۱ ثانیه) که تصویر پرنده را در یک لحظه به وضوح ثبت کند. البته برای تنظیم سرعت شاتر بایست به میزان نور محیط نیز توجه داشت، زیرا سرعت بالای شاتر، میزان نوری که به سنسور برخورد می‌کند را کاهش می‌دهد. بنابراین سرعت‌های بسیار بالا ممکن است برای محیط‌های کم‌نور خیلی مناسب نباشند. در این صورت عکاس می‌تواند از ISO استفاده کند و یا از باز کردن دیافراگم (انتخاب عدد f کوچکتر) برای افزایش نور ورودی استفاده کند. البته باید توجه داشت باز کردن دیافراگم عمق میدان را کاهش می‌دهد و احتمالا بخش‌هایی از پس‌زمینه از فوکوس خارج شده و کمی تار می‌شوند.

.۳
برای ثبت تصویری واضح از یوزپلنگ در حال حرکت با پس زمینه تار (حس سرعت) و نمای وسیع تر از منظره، عکاس باید تنظیمات و تکنیک های زیر را در نظر بگیرد:

سرعت شاتر: برای اصطلاحا منجمد کردن حرکت یوزپلنگ و جلوگیری از تاری، عکاس به سرعت شاتر بالا نیاز دارد. نقطه شروع می تواند ۱/۱۰۰۰ ثانیه یا سریعتر باشد، به خصوص که یوزپلنگ یکی از سریع ترین حیوانات است. با این حال، برای تأکید بر حس سرعت با پس‌زمینه‌ای کمی تار و در عین حال تیز نگه داشتن یوزپلنگ، استفاده از تکنیک panning (حرکت دوربین به صورت افقی همراه با سوژه) با سرعت شاتر کمی آهسته‌تر، مانند ۱/۵۰۰ ثانیه نیز ممکن است مؤثر باشد.

عمق میدان (Depth of Field): برای اینکه فوکوس روی یوزپلنگ واضح باشد و پس‌زمینه تار شود، نیاز به عمق میدان کم است. استفاده از دیافراگم باز (یک عدد f کوچک، مثلاً f/۲.۸ یا f/۴) باعث می‌شود که عمق میدان کاهش یابد و تنها یوزپلنگ به صورت واضح در تصویر ظاهر شود، در حالی که پس‌زمینه تار می‌گردد. همچنین باعث می‌شود نور بیشتری به سنسور برخورد کند و سرعت شاتر سریع‌تر را ممکن کند.

میدان دید (Field of View): برای گنجاندن دید وسیعی از منظره در قاب، باید از یک لنز با فاصله کانونی کوتاه‌تر (زاویه بازتر) استفاده کرد تا میدان دید گسترده‌تری حاصل شود. البته، باید توجه داشت که یوز به عنوان سوژه اصلی باید به اندازه کافی بزرگ و واضح باشد، پس استفاده از یک لنز با فاصله کانونی معقول مانند ۷۰-۲۰۰ میلی‌متر می‌تواند مناسب باشد، بسته به فاصله عکاس از سوژه.

پیشنهادات بیشتر:

ISO: بایست ISO را بر اساس شرایط نوری تنظیم کنیم. اگر نور کافی باشد، ایزو کمتر (مانند ISO ۱۰۰ یا ۲۰۰) می تواند نویز را به حداقل برساند.

تکنیک panning: برای اینکه یوزپلنگ را در فوکوس نگه داریم از این تکنیک که شامل حرکت دوربین همراه با سوژه متحرک با همان سرعت در طول نوردهی است استفاده می‌کنیم. پانینگ کمک می کند تا یوزپلنگ در فوکوس واضح در برابر پس زمینه تار حرکت نگه داشته شود و حس سرعت را تقویت کند.

حالت فوکوس: استفاده از حالت فوکوس خودکار مداوم برای حفظ تمرکز روی یوزپلنگ متحرک بسیار مهم است. اگر دوربین اجازه می‌دهد، انتخاب یک ناحیه فوکوس خودکار پویا یا ردیابی فوکوس خودکار می‌تواند برای حفظ فوکوس یوزپلنگ در حین حرکت مفید باشد.

۵.

الف.

در OpenCV به طور پیش‌فرض تصاویر را در فضای رنگی BGR نمایش می‌دهد اما دیفالت Matplotlib این است که تصاویر در فرمت RGB می‌باشند. بنابراین، اگر تصویری را با OpenCV بخوانیم (که در فرمت BGR خواهد بود) و مستقیماً آن را با استفاده از Matplotlib بدون تبدیل به RGB نمایش دهیم، کانال های قرمز و آبی با هم عوض می‌شوند که عامل تفاوت ذکر شده در سوال است.

ب.

۱. ابتدا با استفاده از تابع flip معکوس تصویر در جهت عمودی را به دست می‌آوریم. سپس با استفاده از vconcat تصویر اولیه و تصویر معکوس شده را به صورت عمودی concat می‌کنیم. در آخر نیز با استفاده از imshow نتیجه را نمایش می‌دهیم.

```
# Invert the image vertically
image_inverted = cv2.flip(I, 0)

# Concatenate the image with its vertical invert
concatenated_image = cv2.vconcat([I, image_inverted])

cv2.imshow('Concatenated Image', concatenated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()  # Close the window
```

۲. در این بخش نیز مراحل بخش قبل را تکرار می‌کنیم تنها با این تفاوت که برای آن که تصویر معکوس شده بالاتر از تصویر اصلی نمایش داده شود، در ورودی تابع vconcat، ابتدا تصویر معکوس و سپس تصویر اصلی را وارد می‌کنیم.

```
# Invert the image vertically
image_inverted = cv2.flip(I, 0)

# Concatenate the inverted image with the original image vertically
concatenated_image = cv2.vconcat([image_inverted, I])

# Display the concatenated image using OpenCV
cv2.imshow('Concatenated Image', concatenated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()  # Close the window
```

۳. کد این بخش با دو بخش پیشین دو تفاوت دارد. ابتدا اینکه آرگومان دوم تابع flip را ۱ می‌دهیم تا تصویر در جهت افقی معکوس شود و دوم اینکه برای concat کردن دو تصویر از تابع hconcat استفاده می‌کنیم تا تصاویر به صورت افقی در کنار هم قرار گیرند.

```
# Invert the image horizontally
image_inverted = cv2.flip(I, 1)

# Concatenate the image with its horizontal invert
concatenated_image = cv2.hconcat([I, image_inverted])

# Display the concatenated image using OpenCV
cv2.imshow('Concatenated Image', concatenated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()  # Close the window
```

۴. در این بخش نیز مراحل بخش قبل را تکرار می‌کنیم تنها با این تفاوت که برای آن که ابتدا تصویر معکوس شده و سپس تصویر اصلی نمایش داده شود، در ورودی تابع hconcat، ابتدا تصویر معکوس و سپس تصویر اصلی را وارد می‌کنیم.

```
# Invert the image horizontally
image_inverted = cv2.flip(I, 1)

# Concatenate the inverted image with the original image horizontally
concatenated_image = cv2.hconcat([image_inverted, I])

# Display the concatenated image using OpenCV
cv2.imshow('Concatenated Image', concatenated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()  # Close the window
```

۵. ابتدا مانند بخش‌های پیشین با استفاده از تابع imread تصویر را می‌خوانیم. سپس بایست x, y, w و h را به صورت مناسب مقداردهی کنیم. با کمی آزمون و خطا مختصات مربوط به یکی از پنجره‌های ساختمان را پیدا کردم.
پس از آن یک ماسک با ابعاد تصویر به صورت سیاه و سفید ساختم و سپس رنگ پیکسل‌ها را در مختصاتی که قبلا پیدا کردم به سفید (۲۵۵) تغییر دادم. پس از آن رنگ مد نظر برای پنجره را ساختم (آبی). سپس با استفاده از np.where هرجا ماسک مقدار ۲۵۵ داشت (که نشان‌دهنده‌ی این است که آن پیکسل داخل پنجره مدنظر قرار دارد.) رنگ آن پیکسل را به رنگ جدید تعریف شده تغییر دادم.
در انتها به کمک تابع imwrite تصویر جدید را سیو کردم.

```
Image = cv2.imread('ComputerDepartment.jpg', cv2.IMREAD_UNCHANGED)
x, y, w, h = 322, 153, 25, 21  # coordinates and size of the window

# Create a mask for the window
window_mask = np.zeros(Image.shape[:2], dtype="uint8")
cv2.rectangle(window_mask, (x, y), (x + w, y + h), 255, -1)

# Define the new color you want for the window, e.g., blue
new_color = [255, 0, 0]  # BGR format for blue

# Change the color of the window using the mask
Image[np.where(window_mask == 255)] = new_color

# Display the result
cv2.imshow('Modified Image', Image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
cv2.imwrite('Colored_Window.jpg', Image)
```

۶. ابتدا با استفاده از تابع آماده split کانال‌های رنگی متفاوت عکس را از یک‌دیگر جدا کرده و هریک را ذخیره می‌کنیم.

```
blue_channel, green_channel, red_channel = cv2.split(iust)
```

حالا برای اینکه بخواهیم یک کانال خاص را نمایش دهیم، لازم است مقدار سایر کانال‌ها را صفر کنیم. برای مثال برای نمایش کانال آبی، ابتدا یک کانال با مقادیر ۰ و به ابعاد کانال‌های تصویر می‌سازیم، سپس با استفاده از تابع merge کانال‌های سبز و قرمز را صفر می‌کنیم و کانال آبی را نگه می‌داریم. برای سایر کانال‌ها نیز به طریق مشابه عمل می‌کنیم.

```
# For the blue image: keep the blue channel and set the others to zero
blue_image = cv2.merge([blue_channel, zero_channel, zero_channel])
cv2.imshow('Blue Channel in Blue', blue_image)

# Wait for any key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

تحلیل:

- پس‌زمینه سفید: از آن‌جا که رنگ سفید هر سه رنگ آبی، سبز و قرمز را درون خود دارد، اگر دو کانال را صفر کنیم و یک کانال را حفظ کنیم پس‌زمینه سفید تصویر به رنگ کانال حفظ شده درمی‌آیند.
- نوشته سیاه: درمورد رنگ سیاه، از آن‌جا که این رنگ در هر سه کانال RGB مقدار ناچیزی (تقریبا صفر) دارد، با صفر کردن دو کانال نیز تغییر قابل توجهی نمی‌کند و سیاه می‌ماند.
- بدنه آبی: در رنگ آبی کانال‌های قرمز و سبز مقدار ناچیز (حدود صفر) دارند و کانال آبی است که مقدار بالایی دارد. بنابراین در تصاویری که کانال قرمز یا سبز را حفظ کرده‌ایم، آبی به سیاه متمایل شده است، زیرا رنگ آبی از ابتدا مقدار R و G نزدیک به صفر داشته و ما هم مقدار B را صفر کرده‌ایم پس رنگ تقریبی سیاه حاصل می‌شود. اگر کانال آبی را حفظ کنیم تغییر خاصی در رنگ آن ایجاد نمی‌شود چون R و G صفر شده‌اند که از ابتدا نیز مقدارشان حدودا همین بوده است و فقط رنگ آبی دیده می‌شود.
- شعله قرمز: در رنگ قرمز کانال‌های آبی و سبز تقریبا صفر هستند و تنها کانال قرمز است که مقدار دارد. بنابراین اگر مقدار این کانال صفر شود هر سه کانال صفر هستند و رنگ سیاه نمایش داده می‌شود. اما اگر کانال قرمز حفظ و دو کانال دیگر صفر شوند تغییر خاصی در رنگ شعله ایجاد نمی‌شود و قرمز می‌ماند.

منابع مورد استفاده در سوالات:

https://gemini.google.com/app

https://chrisbrayphotography.com/tips/birds_in_flight.php

https://www.nickdalephotography.com/blog/shutter-speeds-for-action-shots