



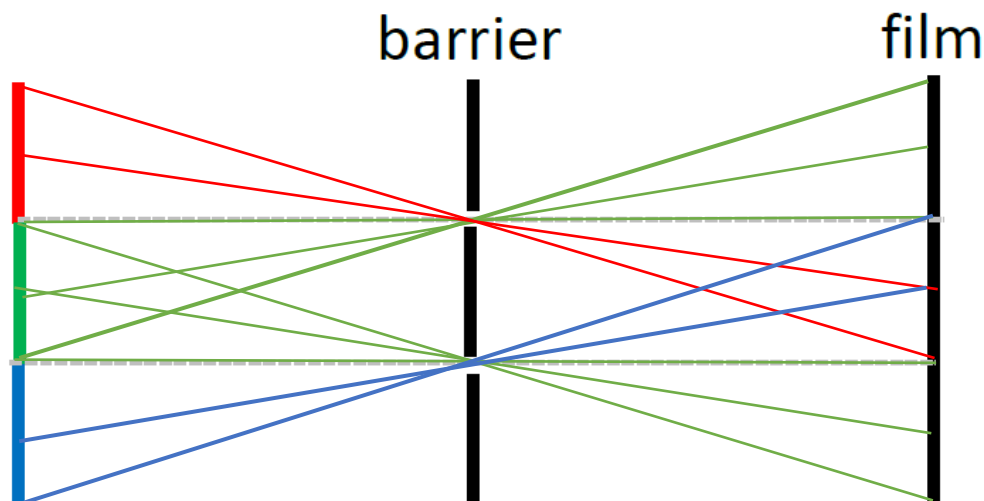
## گزارش تمرین دوم بینایی ماشین

نام تهیه کننده: ملیکا نوبختیان

شماره دانشجویی: ۹۷۵۲۲۰۹۴

نسخه: ۱

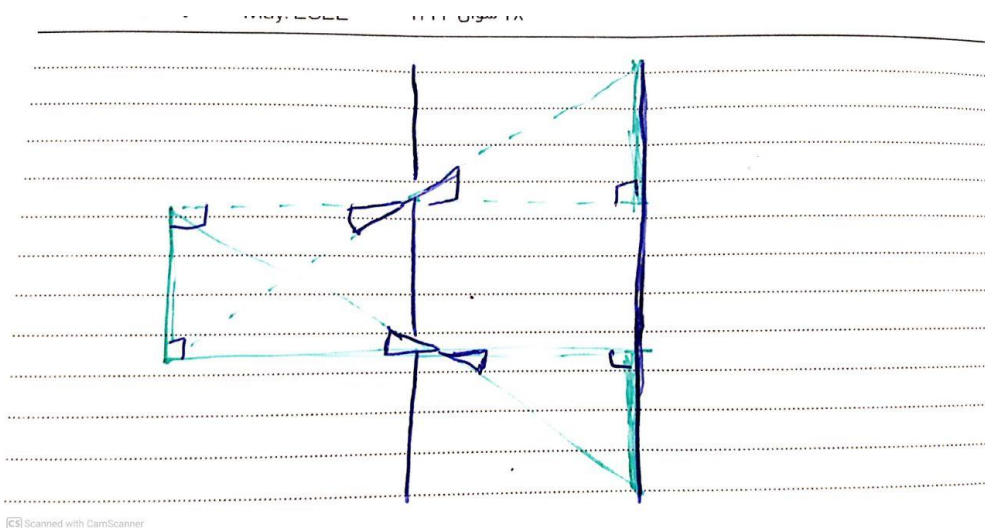
## ۱- سوال اول



در دوربین pinhole اگر دریچه‌ها بزرگ شوند، نور منعکس شده در بخش بیشتری از تصویر تاثیر خواهد گذاشت و تصویر تار خواهد شد. اما اگر دریچه کوچک شود تار شدن را کاهش می‌دهد اما میزان نور ورودی به دوربین را کم می‌کند. هم چنین باعث پراکندگی نور نیز خواهد شد.

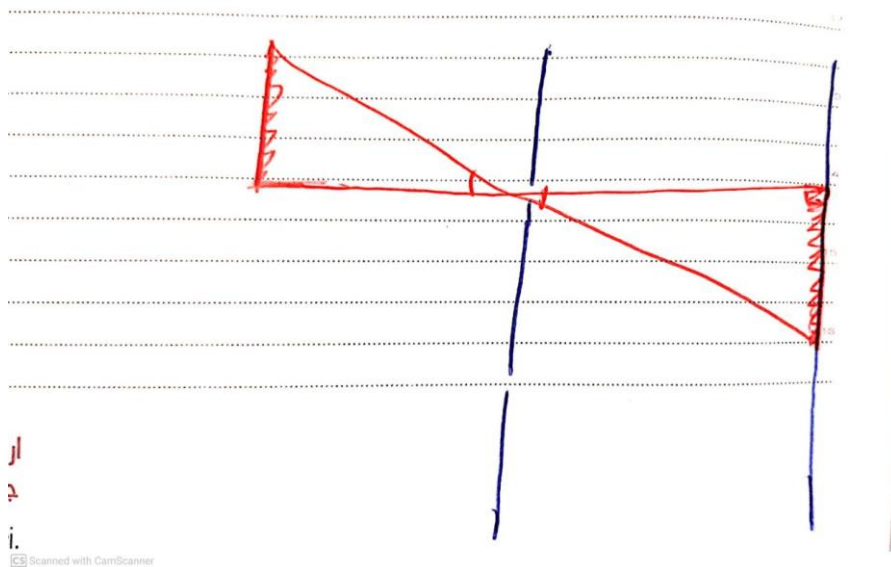
در دوربین pinhole هر دریچه یک تصویر متفاوت به ما می‌دهد و هر چه تعداد دریچه‌ها بیشتر باشد تصاویر بیشتر خواهند شد. اما این تصاویر ممکن است در قسمت‌هایی هم پوشانی داشته باشند و ترکیب شوند مثل چیزی که در دوربین pinhole سوال و تصویر بالا اتفاق می‌افتد که با تشابه مثلث‌ها بیشتر با تصویری که تشکیل می‌شود آشنا خواهیم شد.

در مورد قسمت سبز جسم به این شکل خواهیم داشت:



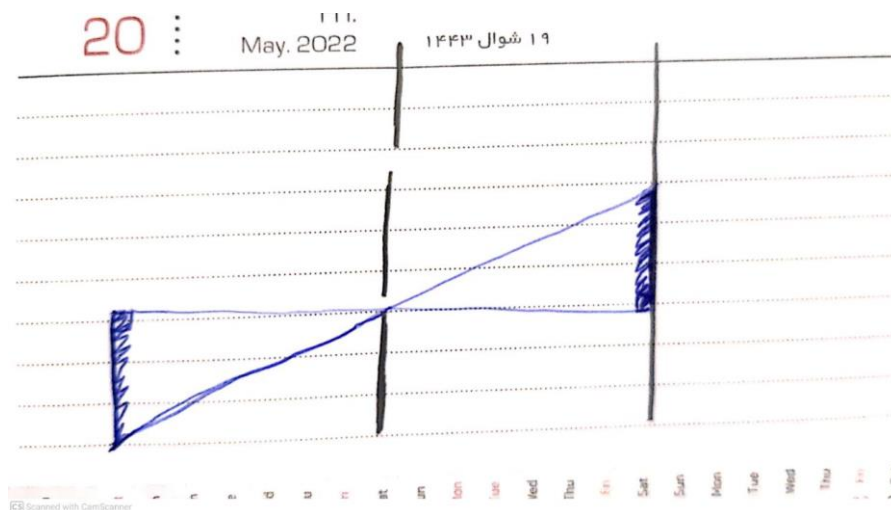
با تشابه مثلث‌ها واضح است که دو تکه بالا و پایین فیلم به شکل بالا سبز خواهند شد.

در مورد قسمت قرمز خواهیم داشت:



تنها پرتوهایی که از قسمت قرمز منعکس می‌شوند و از دریچه اول عبور می‌کنند روی فیلم می‌افتند و در قسمت وسط فیلم قرار می‌گیرند.

در نهایت قسمت آبی نیز به شکل زیر خواهد بود:



تنها نورهایی که از قسمت آبی منعکس می‌شوند و از دریچه دوم عبور می‌کنند روی فیلم می‌افتند. نور قسمت آبی هم درست مانند قسمت قرمز در وسط می‌افتد.

با توجه به این موارد تصویری که در نهایت خواهیم داشت در قسمت بالا و پایین سبز خواهد بود و در قسمت وسط با توجه به اینکه نورهای قرمز و آبی ترکیب می‌شوند، رنگ ارغوانی خواهد داشت.

## ۲- سوال دوم

این تصویر با استفاده از دوربین لنزدار ثبت شده است زیرا در این نوع دوربین فرمول زیر را داریم:

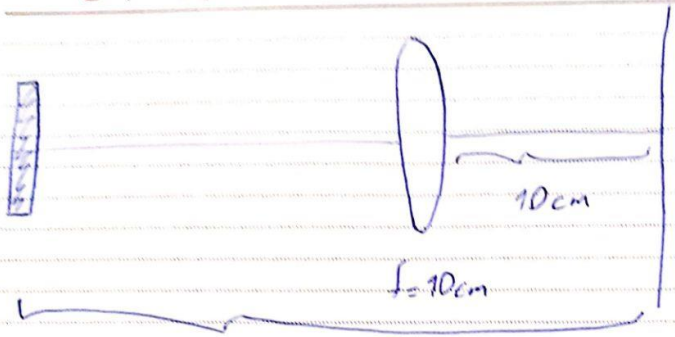
$$\frac{1}{f} = \frac{1}{v} + \frac{1}{u}$$

در این فرمول  $f$  فاصله کانونی لنز،  $u$  فاصله جسم تا لنز و  $v$  فاصله لنز تا پرده است. اگر پرده را در یک جای ثابت در نظر بگیریم و لنز هم ثابت باشد به عبارتی دیگر  $v$  ثابت باشد، تنها نقاطی از جسم که در آن فاصله  $u$  مشخص که در فرمول قرار می‌گیرد و پاسخ درست را با توجه به  $v$  و  $f$  مشخص به ما می‌دهند، تصویر واضح و درستی خواهند داشت. نقاطی که دورتر یا نزدیک‌تر از این فاصله باشند تصویرشان به درستی روی پرده قرار نخواهد گرفت و بنابراین تصویری تار خواهند داشت که مشابه چیزی است که در تصویر سوال مشاهده می‌کنیم.

قسمت وسط متن که واضح است در فاصله  $u$  درستی از لنز قرار گرفته به این دلیل تصویر واضحی دارد ولی پایین متن و بالای متن به ترتیب در فاصله  $u$  نزدیک‌تر و  $u$  دورتر قرار گرفته‌اند و به این دلیل تصویر واضحی ندارند. به این خاصیت عمق میدان می‌گوییم که در دوربین‌های لنزدار اتفاق می‌افتد.

برای بهبود تصویر کاری که می‌توانیم انجام دهیم این است که علاوه بر لنز از دریچه نیز استفاده کنیم. زیرا این کار باعث می‌شود تصویر واضح‌تری داشته باشیم و نقاط تار کمتر شوند.

### ۳- سوال سوم



$$u + v = 40 \text{ cm} \quad u = 40 - v$$

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v} \quad \frac{1}{10} = \frac{1}{u} + \frac{1}{40 - u}$$

$$\frac{1}{10} = \frac{40}{40u - u^2} \quad u^2 - 40u + 400 = 0$$

$$u = u = 20 \text{ cm}$$

برای اینکه تصویر واضح باشد باید فاصله جسم تا لنز ۲۰ سانتی‌متر باشد. فاصله لنز تا پرده ۲۰ سانتی‌متر است. فاصله جسم تا پرده ۴۰ سانتی‌متر است.

اردیبهشت  
جمعه  
Fri.  
May 2022  
۵ شوال ۱۴۴۳

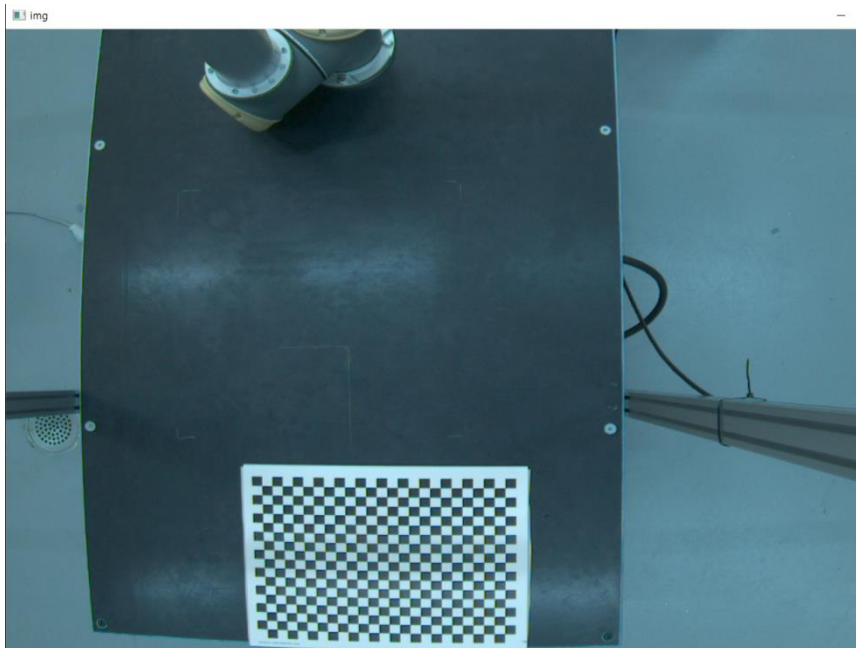
## ۴- سوال چهارم

تصویر img1.png را به شکل زیر می خوانیم و ابعاد آن را به  $1080 \times 720$  تغییر می دهیم و سپس آن را نشان

می دهیم:

```
import cv2
imm = cv2.imread('images/img1.png', cv2.IMREAD_COLOR)
imm = cv2.cvtColor(imm, cv2.COLOR_BGR2RGB)
cv2.imshow('img', imm)
cv2.waitKey(0)
```

تصویر به شکل زیر خواهد بود:



ما می توانیم از یک با لنزهای مختلف های عکس بگیریم که هر کدام میدان دید مخصوص به خود را دارند و مقدار بیشتر یا کمتری از یک صحنه را پوشش می دهند. اگر از لنزهای wide استفاده کنیم و نزدیک به جسم یا صحنه باشیم میدان دید بیشتری خواهیم داشت ولی اگر از لنز narrow استفاده کنیم چون زاویه دید کم است باید در فاصله دورتری بایستیم و دید محدودتری خواهیم داشت. اما کدام یک از این دو انتخاب تصویر بهتری به ما خواهد داد؟ هنگامی که در فاصله دورتر با لنز narrow تصویر را می گیریم، چون عمق نقاط جسم نسبت به هم تفاوت زیادی ندارند تصویر بهتر و واقعی تری خواهیم داشت. اما اگر تصویر با لنز wide و در فاصله کم ثبت شود، عمق نقاط یک جسم نسبت به هم تفاوت قابل توجهی خواهند داشت و به این دلیل تصویر غیرواقعی و تا حدودی کارتونی جلوه خواهد کرد. به این اتفاق اعوجاج تصویر می گوئیم.

چون بعضی از توابعی که در ادامه با آن ها کار داریم نیاز به تصویر gray scale دارند، در ابتدا تصویر را به gray scale تبدیل کنیم:

```
# convert to gray scale
imm_gray = cv2.cvtColor(imm_rgb, cv2.COLOR_RGB2GRAY)
```

برای استفاده از تابع `findChessboardCorners` باید ابعاد صفحه شطرنج داخلی را بدانیم که ابعاد ما در اینجا  $24 \times 17$  است. پس `pattern` ما به این اندازه خواهد بود:

```
# Chessboard size
pattern = (24, 17)
```

پارامترهای ضروری برای این تابع خود تصویر و اندازه الگو موردنظر ما هستند و خروجی‌های تابع که در قطعه کد زیر می‌بینید `ret` و `corners` هستند که `ret` نشان می‌دهد آیا در تصویر ما صفحه شطرنجی با ابعاد داده شده وجود داشته است یا نه و در صورت وجود `corners` مختصات تمام گوشه‌ها را می‌دهد:

```
ret, corners = cv2.findChessboardCorners(img_gray, pattern)
```

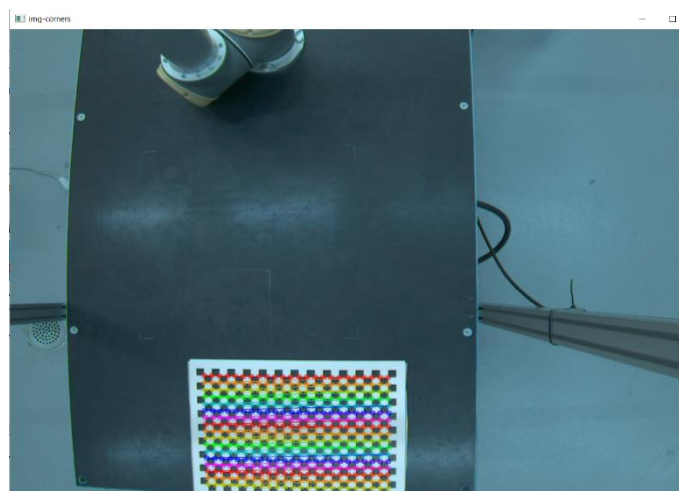
برای اینکه بتوانیم مختصات دقیق‌تری از نقاط گوشه داشته باشیم، از تابع `cornerSubPix` استفاده می‌کنیم. ورودی‌های این تابع به این صورت هستند: خود تصویر، `corner` هایی که در تابع قبل آن‌ها را به دست آوردیم، `winSize` که نصف `searchWindow` ما برای یافتن نقاط `corner` است. من مقدار  $12 \times 12$  را برای این کار در نظر گرفتم تا مساحتی بیشتر از کل صفحه شطرنج را نیز در بر بگیرد. پارامتر بعدی `zeroZone` است که نصف اندازه `deadZone` ما را برای جستجو مشخص می‌کند. در صورت تنظیم آن به چیزی غیر از  $(-1, -1)$  از وسط پنجره جستجو مقدار پنجره `zeroZone` از آن `exclude` خواهد شد. پارامتر آخر نیز `criteria` است که مشخص می‌کند فرآیند `corner refinement` در چه صورتی به اتمام خواهد رسید. این فرآیند در صورتی به پایان خواهد رسید که یا به `criteria.maxCount` در `iteration` ها برسیم یا در یک `iteration` مشخص مکان یک `corner` کمتر از `criteria.epsilon` تغییر کند. در اینجا تعداد `iteration` حداکثری را برابر ۳۰ و مقدار `epsilon` را برابر ۰.۱ قرار می‌دهیم:

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.1)
acc_corners = cv2.cornerSubPix(img_gray, corners, (12, 12), (-1, -1),
criteria)
```

حالا با استفاده از `drawChessboardCorners` نقاط پیدا شده را رسم می‌کنیم. به این تابع تصویر `rgb` اصلی، اندازه الگو صفحه شطرنج، `corner` های با مختصات دقیق از `cornerSubPix` و خروجی `ret` را که قبلا به دست آوردیم را به آن می‌دهیم:

```
cv2.drawChessboardCorners(img_rgb, pattern, acc_corners, ret)
```

تصویر خروجی به شکل زیر خواهد بود:



حالا می‌توانیم به سراغ کالیبره کردن دوربین برویم. دو پارامتر مهم که تابع `calibrateCamera` می‌گیرد `objectPoints` و `imagePoints` هستند. `imagePoints` در واقع همان `corner` هایی هستند که تا به حال به دست آورده‌ایم. این نقاط دوبعدی نشان دهنده مختصات `corner` های جدول در تصویر هستند. اما `objectPoints` مختصات سه بعدی یک صفحه شطرنجی با ابعاد الگو را نشان می‌دهد. بعد سوم که عمق را نشان می‌دهد در اینجا برای ما صفر است چون نقاط همگی در یک عمق قرار دارند. به صورت زیر `objpoints` را می‌سازیم:

```
objp = np.zeros((pattern[0] * pattern[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:pattern[0], 0:pattern[1]].T.reshape(-1, 2)
```

که `objpoints` در نهایت به صورت زیر خواهد بود:

```
[[ 0.  0.  0.]
 [ 1.  0.  0.]
 [ 2.  0.  0.]
 ...
 [21. 16.  0.]
 [22. 16.  0.]
 [23. 16.  0.]]
```

در نهایت به صورت زیر دوربین را کالیبره می‌کنیم:

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
imm gray.shape, None, None)
```

`dist` در خود پارامترهای مربوط به `distortion` را خواهد داشت که به صورت زیر خواهد بود:

```
k1: -0.3498735136075164, k2: -4.867966736041907, p1: -0.0037981521696158708, p2: 0.09071582118274732, k3: 22.370674256508806
```

در قدم بعد می‌خواهیم اعوجاج تصویر `img5.png` را با استفاده از این پارامترها حذف کنیم. یکی از پارامترهایی که کالیبره کردن به ما داده‌است `mtx` است که ماتریکس دوربین است اطلاعاتی در مورد `focal length` و `optical centers` را به ما می‌دهد. برای اینکه بتوانیم با توجه به تصویرمان به ماتریکس `optimal` تری برسیم از تابع `getOptimalNewCameraMatrix` استفاده می‌کنیم. برای این کار به طول و عرض تصویرمان نیاز داریم. هم چنین یک پارامتر دیگر این تابع `alpha` است که اگر برابر صفر باشد سعی می‌کند یک سری از پیکسل‌های ناخواسته را حذف کند پس ممکن است بعضی از پیکسل‌ها حذف شوند ولی اگر برابر یک باشد تمام پیکسل‌ها را حفظ خواهد کرد:

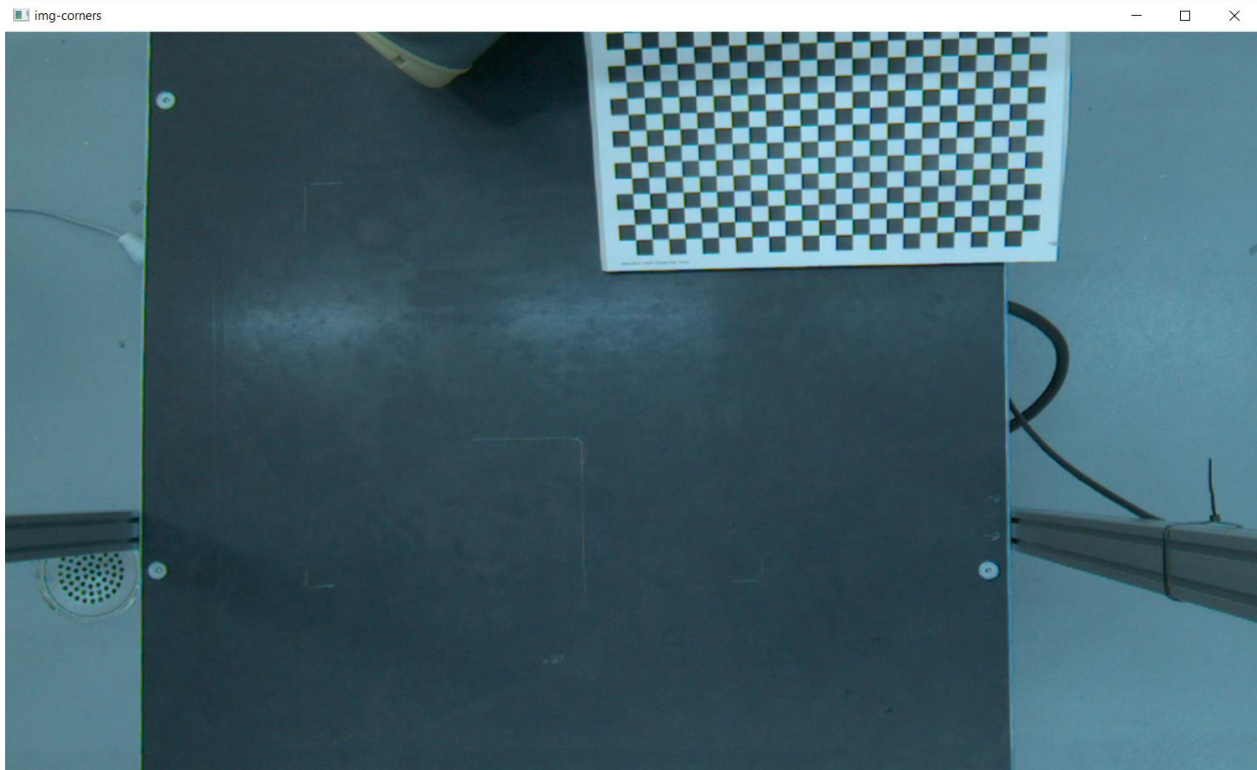
```
img = cv2.imread('images/img5.png', cv2.IMREAD_COLOR)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
h, w = img_rgb.shape[:2]
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w, h), 1, (w, h))
```

خروجی‌های این تابع ماتریکس جدید دوربین و `roi` خواهند بود. `Roi` به طور کلی تمام محدود پیکسل‌های خوب در یک ناحیه `undistorted` را نشان می‌دهد که به ما در `crop` کردن تصویر کمک می‌کند. سپس با کمک تابع `undistort` اعوجاج تصویر را حذف می‌کنیم. سپس با استفاده از پارامترهای به دست آمده از `roi` تنها نواحی مطلوب تصویر را جدا می‌کنیم:



```
# undistort
dst = cv2.undistort(img_conv, mtx, dist, None, newcameramt)
# crop the image
x, y, w, h = roi
dst = dst[y:y + h, x:x + w]
cv2.imshow('img-corners', dst)
cv2.waitKey(0)
```

تصویر به شکل زیر خواهد شد:



حالا با استفاده از تصاویر ۱ تا ۴ سعی می‌کنیم دوربین را کالیبره کنیم و اعوجاج تصویر ۵ را حذف کنیم:

```
# calibrate with img1 to img4
new_objpoints = []
new_imgpoints = []
images = glob.glob('images/*.png')
curr_im = None
for fname in images[:-1]:
    print(fname)
    img = cv2.imread(fname, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    curr_im = gray
    ret, corners = cv2.findChessboardCorners(gray, pattern)
    if ret:
        new_objpoints.append(objp)
        acc_corners = cv2.cornerSubPix(gray, corners, (12, 12), (-1, -1),
criteria)
        new_imgpoints.append(acc_corners)
        cv2.drawChessboardCorners(img, pattern, acc_corners, ret)
        cv2.imshow('img', img)
        cv2.waitKey(0)

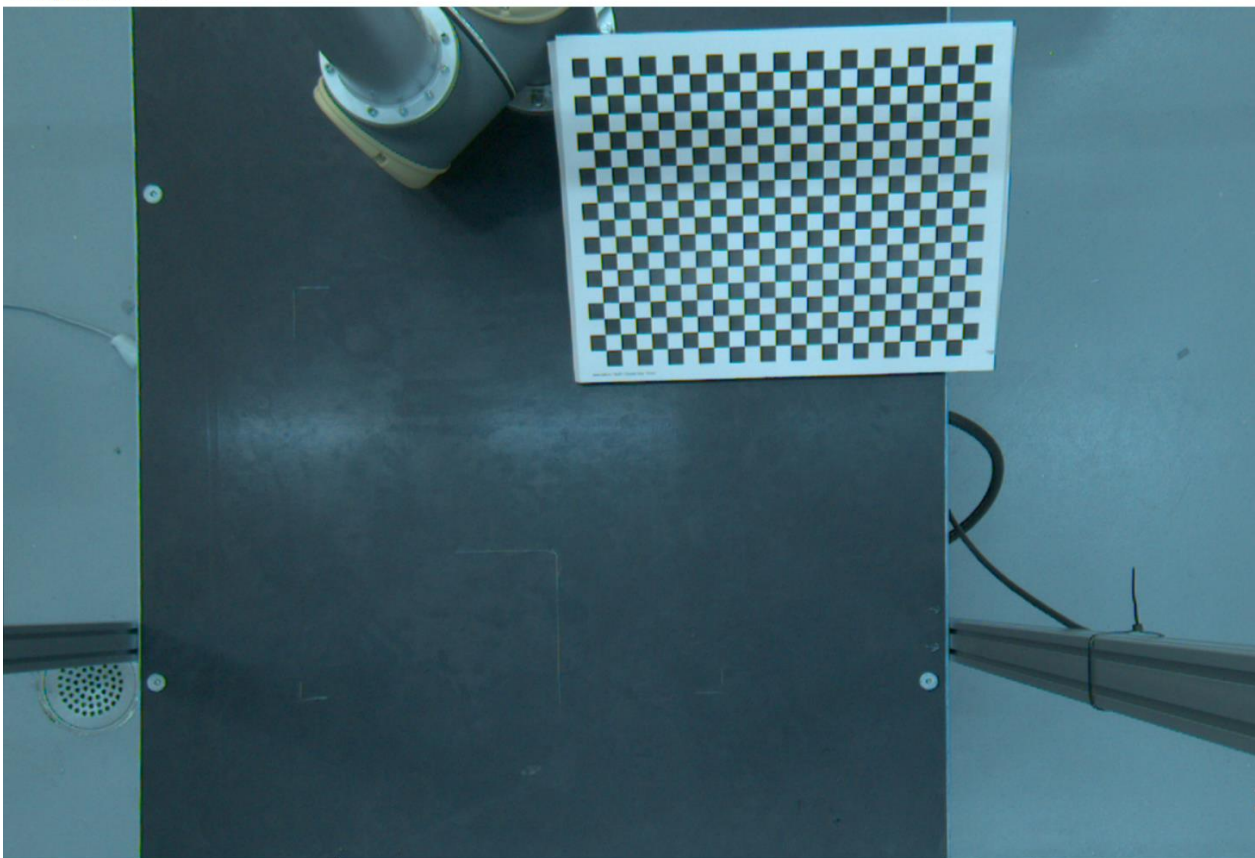
img = cv2.imread('images/img5.png', cv2.IMREAD_COLOR)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```



```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(new_objpoints,
new_imgpoints, curr_im.shape, None, None)
h, w = img_rgb.shape[:2]
newcameramt, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w, h), 1, (w,
h))
# undistort
dst = cv2.undistort(img_rgb, mtx, dist, None, newcameramt)
# crop the image
x, y, w, h = roi
dst = dst[y:y + h, x:x + w]
cv2.imshow('img5_with_undistort_all', dst)
cv2.waitKey(0)
```

تصویر نهایی به شکل زیر خواهد بود:

img5\_with\_undistort\_all



در تصویر دوم که اعوجاج آن حذف شده است چون از تصاویر بیشتری برای کالیبره کردن دوربین استفاده شده بود اعوجاج به طور بهتری حذف شده است و شاهد خطوط صاف تری در این تصویر نسبت تصویری که تنها با یک تصویر undistort شده بود هستیم. تعداد تصاویر بیشتر و الگوهای بهتر می تواند به ما کمک کند تا به شکل بهتری اعوجاج را حذف کنیم.

منابع:

[https://docs.opencv.org/4.x/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html)