ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO BÀI TẬP XỬ LÝ ẢNH

Project 1.1:

XÁC ĐỊNH KÍCH THƯỚC BÀN CỜ

Giảng viên hướng dẫn: TS. Nguyễn Thị Oanh

Sinh viên: Nguyễn Trần Hiếu Giang 20191804

Phùng Phú Cường 20190084

Nguyễn Đức Hiếu 20192845

HÀ NỘI, 06/2023

MỤC LỤC

I. Giải thích code 2
1. Xử lý ảnh 2
a) Hàm lọc nhiễu sin 2
b) Hàm xử lý ảnh 2
c) Tải ảnh lên để xử lý 3
2. Bộ lọc đường kẻ không hợp lệ 3
3. In kết quả và độ chuẩn xác 5
4. Thể hiện trên ảnh gốc 5
II. Kết quả và đánh giá 7
1. Ảnh Chessboard_0451 7
2. Ảnh Chessboard_0451_1 7
3. Ảnh Chessboard_0451_2 8
4. Ảnh Chessboard_0481 8
5. Ảnh Chessboard_0481_0 9
6. Ảnh Chessboard_0481_1 9
7. Ảnh Chessboard_0511 10
8. Ảnh Chessboard_0541 10
9. Ảnh Chessboard_0601 11
10. Ảnh Chessboard_0631 11

I. Giải thích code

1. Xử lý ảnh

a) Hàm lọc nhiễu sin

Thực hiện biến đổi Fourier 2D (np.fft.fft2) trên ảnh đen trắng (được biến đổi ở hàm xử lý ảnh bên dưới). Biến đổi này sẽ chuyển đổi ảnh từ miền không gian sang miền tần số. Dùng np.fft.fftshift để dịch chuyển thành phần tần số 0 về trung tâm của phổ tần số, giúp cho việc xử lý và phân tích tần số trở nên dễ dàng hơn.

```
def sine_denoise(image):
    # Compute the 2D Fourier transform of the image
    fourier_transform = np.fft.fft2(image)
    # Shift the zero-frequency component to the center of the spectrum
    fourier_shift = np.fft.fftshift(fourier_transform)
```

Tạo một mặt nạ để áp dụng lên dữ liệu tần số. Mặt nạ này được tạo bằng cách tạo hai đường thẳng trắng trên nền đen.

```
# Create a mask with two lines
mask1 = np.ones_like(image) * 255
mask2 = np.zeros_like(image)
center_y = mask1.shape[0] // 2
center_x = mask1.shape[1] // 2
var = cv2.line(mask2, (center_x + 8, center_y), (center_x + 8, center_y), (255, 255, 255), 1)[0]
var2 = cv2.line(mask2, (center_x - 8, center_y), (center_x - 8, center_y), (255, 255, 255), 1)[0]
mask = mask1 - mask2
```

Ta lấy mask và chia cho 255 để tạo ra một phiên bản dữ liệu tần số đã được lọc. Giới hạn các giá trị của ảnh tái tạo trong khoảng từ 0 đến 255 và chuyển đổi nó thành định dạng uint8 để đảm bảo rằng ảnh có độ sáng và độ tương phản phù hợp.

```
# Apply the mask to the shifted Fourier transform
dft_shift_masked = np.multiply(fourier_shift, mask) / 255
# Shift the origin back to the upper left corner
back_ishift_masked = np.fft.ifftshift(dft_shift_masked)
# Compute the inverse Fourier transform to obtain the filtered image
img_filtered = np.fft.ifft2(back_ishift_masked)
# Clip and convert the image back to uint8 format
img_filtered = np.abs(img_filtered).clip(0, 255).astype(np.uint8)
return img_filtered
```

b) Hàm xử lý ảnh

Ta bắt đầu quá trình xử lý ảnh với chuyển ảnh đang có thành ảnh đen trắng thông qua việc sử dụng hàm cv2.cvtColor với setting cv2.COLOR_BGR2GRAY. Tiếp đó, làm mờ ảnh đã được lọc nhiễu sin sử dụng hàm Gaussian Blur và sử dụng hàm Canny để phát hiện các đường biên:

```
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Blur the image for better edge detection
img_blur = cv2.GaussianBlur(sine_denoise(gray), (5, 5), 0)
# Apply the Canny edge detector to find edges in the image
edges = cv2.Canny(img_blur, 30, 55, apertureSize=3)
```

Các đường biên tìm được trên ảnh thường rất mỏng và có dấu hiệu bị đứt đoạn (fractured), do đó ta tiếp tục sử dụng tiến trình Closing (Giãn – Dilation và Co – Erosion) để những đường biên này liền mạch hơn và rõ hơn; thuận tiện cho quá trình tìm các đường thẳng trên ảnh:

```
# Close the image with Dilation and Erosion to connect fractured lines
kernel = np.ones((2, 2), np.uint8)
dilate = cv2.dilate(edges, kernel, iterations=2)
ero = cv2.erode(dilate, kernel, iterations=1)
```

Với những ảnh bị nhiễu muối tiêu, sau giai đoạn xử lý trên, ta sẽ thu được rất nhiều điểm trắng gây nhiễu trên ảnh. Do đó, ta tính số pixel đen và trắng trên ảnh, nếu số pixel trắng thu được nhiều hơn số pixel đen, ta có thể khẳng định ảnh bị nhiễu muối tiêu. Khi đó, ta sẽ thêm hàm xử lý muối tiêu NlMeansDenoising và sử dụng hàm Median Blur để làm mờ ảnh hiệu quả hơn:

```
# Calculate Black and White pixels to detect if the original image  has salt and pepper noise
number_of_white_pix = np.sum(ero == 255)
number_of_black_pix = np.sum(ero == 0)
if number_of_black_pix < number_of_white_pix:
    noiseless_image = cv2.fastNlMeansDenoisingColored(image, None, 20, 20, 7, 21)
    noiseless_gray = cv2.cvtColor(noiseless_image, cv2.COLOR_BGR2GRAY)
    noiseless_blur = cv2.medianBlur(noiseless_gray, 3)
    noiseless_edges = cv2.Canny(noiseless_blur, 30, 55, apertureSize=3)
    noiseless_dilate = cv2.dilate(noiseless_edges, kernel, iterations=2)
    noiseless_ero = cv2.erode(noiseless_dilate, kernel, iterations=1)
```

c) Tải ảnh lên để xử lý

Ảnh được tải lên thông qua hàm cv2.imread và xử lý bằng hàm process:

```
# Load and process the image
img = cv2.imread('Images/Chessboard_0451_2.png')
processed = process(img)
```

Ta sử dụng Hough Lines transform để tìm đường thẳng trong ảnh:

```
# Apply the Hough transform to find lines in the image
lines = cv2.HoughLines(processed, 1, np.pi / 180, 180)
```

Khi sử dụng Hough Lines, ta thu được rất nhiều đường thẳng không hợp lệ trên ảnh: là những đường thẳng ở cùng 1 vị trí, hoặc đường chéo (trên bàn cờ, ta chỉ tìm đường thẳng nằm ngang và đường thẳng nằm dọc). Vì vậy, ta cần thêm 1 hàm để lọc những đường kẻ này.

2. Bộ lọc đường kẻ không hợp lệ

Khởi tạo các mảng để kiểm tra và phân loại, cũng như biến chạy theo số đường thẳng tối đa.

```
filtered_lines = []
check = []
check_horizon = []
check_vertical = []
i = 0
```

Những đường thẳng tìm được bằng biến đổi Hough có chứa 2 giá trị rho (khoảng cách so với các trục) và theta (góc quay); ta chuyển đơn vị của theta từ radian thành độ. Với những đường thẳng có góc quay 180 độ (những đường dọc này có chiều từ trên xuống), ta quy chuẩn về góc 0 độ để thuận tiện cho quá trình lọc. Ngoài ra, ta khởi tạo biến exist để bỏ qua những đường thẳng có cùng vị trí với những đường thẳng đã tìm được. Với những đường thẳng có góc quay gần 45 độ hoặc 135 độ, ta coi đó là đường chéo và bỏ qua.

```
for line in lines:
    exist = False
    rho, theta = line[0]
    # Convert line offset and angle to positive integer and  degree
    rho_c = abs(int(rho))
    theta_c = int(theta / np.pi * 180)
    if 150 <= theta_c <= 210:
        theta_c = abs(theta_c - 180)
    # Skip diagonal lines
    if (30 < theta_c < 75) or (105 < theta_c < 150):
        i += 1
        exist = True
```

Đường thẳng đầu tiên tìm được luôn là đường thẳng đúng (nếu là đường chéo thì sẽ bị bỏ qua ở trên), nên ta đưa vào mảng để kiểm tra trùng lặp.

```
# Catch first line detected since it is always correct
if i == 0:
    check.append((theta_c, rho_c))
    if check[-1][0] > 45:
        check_horizon.append(check[-1])
    else:
        check_vertical.append(check[-1])
    filtered_lines.append(lines[i])
    i += 1
```

Với những đường thẳng có góc quay và khoảng cách so với trục tương tự nhau, ta coi đó là đường thẳng trùng lặp và bỏ qua. Với những đường thẳng hợp lệ, ta sẽ phân loại đó là đường thẳng dọc hay đường thẳng ngang dựa vào góc quay.

```
else:
    # Remove lines at the same position (Offset and Angle similar to each other)
    for check_line in check:
        if abs(theta_c - check_line[0]) <= 3 and abs(rho_c - check_line[1]) <= 16:
            i += 1
            exist = True
            break
    # Detect if line is vertical or horizontal
    if not exist:
        check.append((theta_c, rho_c))
        filtered_lines.append(lines[i])
        i += 1
        if check[-1][0] > 45:
            check_horizon.append(check[-1])
        else:
            check_vertical.append(check[-1])
        check_horizon = sorted(check_horizon, key=lambda last: last[-1])
        check_vertical = sorted(check_vertical, key=lambda last: last[-1])

if i == len(lines):
    break
```

Nếu biến i đạt giá trị bằng số đường thẳng tối đa, tức là ta đã xét tất cả đường thẳng và có thể thoát vòng lặp.

3. In kết quả và độ chuẩn xác

Với những ảnh đã cho, ta tìm được cả hai đường thẳng ngang ở viền trên và viền dưới của bàn cờ, và không tìm được viền trái và viền phải của bàn cờ. Do đó, ta tính số hàng và cột tìm được như dưới. Độ chuẩn xác được tính thông qua số hàng và cột chênh lệch so với bàn cờ gốc.

```
# Calculate the number of rows and columns on the chessboard
rows = len(check_horizon) - 1
cols = len(check_vertical) + 1

print(f'The chessboard size is {rows}x{cols} ({rows} rows and {cols} columns)')
rows_del = abs(rows - 8)
cols_del = abs(cols - 12)
accuracy = 100 - (rows_del/8 + cols_del/12)/2 * 100
print(f'Accuracy: {accuracy}%')
```

4. Thể hiện trên ảnh gốc

Ta vẽ các đường thẳng hợp lệ tìm được sau khi đã lọc lên ảnh gốc, sử dụng các màu khác nhau để dễ nhìn hơn.

```python
# Draw lines on the image to show the grid of the chessboard
i = 0
# Red, Orange, Yellow, Green, Blue
rgb = [(0, 0, 255), (0, 127, 255), (0, 255, 255), (0, 255, 0), (255, 0, 0)]

for line in filtered_lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    x1 = int(x0 + 1000 * (-b))
    y1 = int(y0 + 1000 * a)
    x2 = int(x0 - 1000 * (-b))
    y2 = int(y0 - 1000 * a)

    cv2.line(img, (x1, y1), (x2, y2), rgb[i], 2)
    i += 1
    if i == len(rgb):
        i = 0

# Display the image with lines drawn on it
cv2.imshow('Image with lines', img)
cv2.waitKey(0)
```

II. Kết quả và đánh giá

Tất cả ảnh đã được đặt tên lại để thuận tiện cho quá trình tải ảnh lên và xử lý.

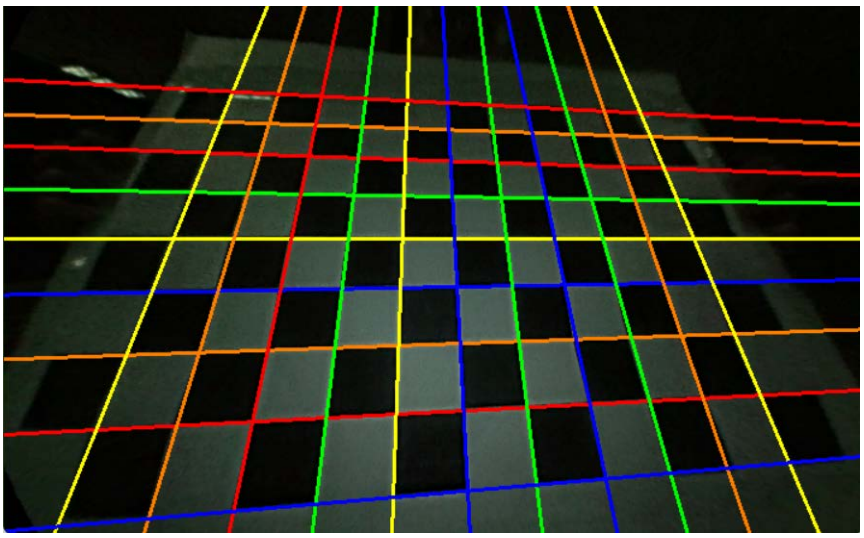
1. Ảnh Chessboard_0451

Kết quả:

```
The chessboard size is 8x12 (8 rows and 12 columns)
Accuracy: 100.0%
```

Nguyên nhân: Tìm được tất cả các đường thẳng hợp lệ.

Đường thẳng tìm được trên ảnh:



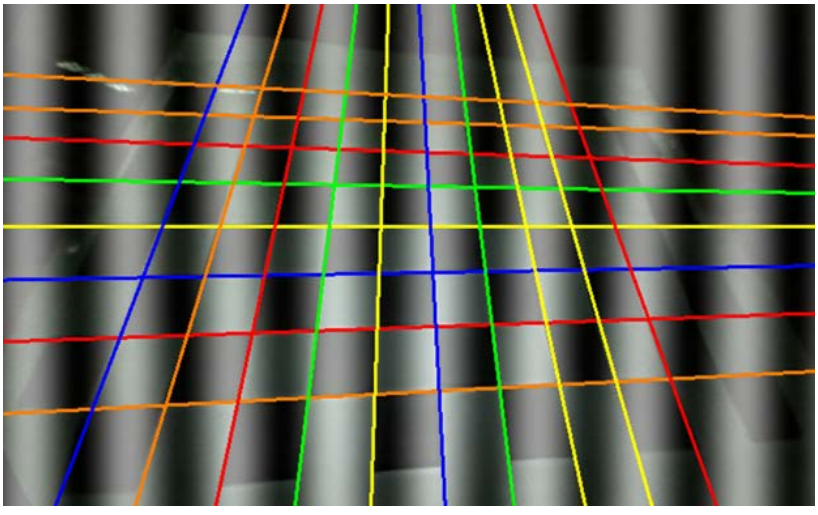
2. Ảnh Chessboard_0451_1

Kết quả:

```
The chessboard size is 7x11 (7 rows and 11 columns)
Accuracy: 89.58333333333333%
```

Nguyên nhân: Không tìm được đường ngang ở viền dưới của bàn cờ, cũng như đường dọc ở sát phải bàn cờ.

Đường thẳng tìm được trên ảnh:



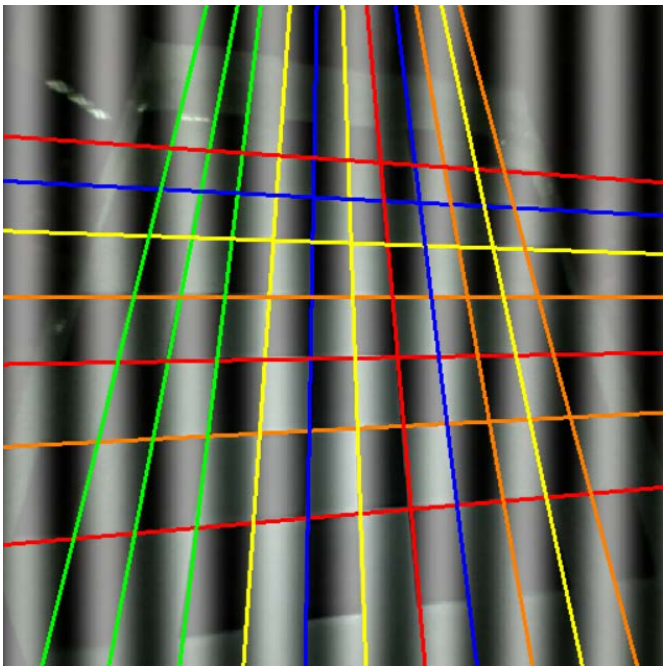
3. Ảnh Chessboard_0451_2

Kết quả:

```
The chessboard size is 6x12 (6 rows and 12 columns)
Accuracy: 87.5%
```

Nguyên nhân: Không tìm được 2 đường ngang ở viền trên và viền dưới của bàn cờ.

Đường thẳng tìm được trên ảnh:



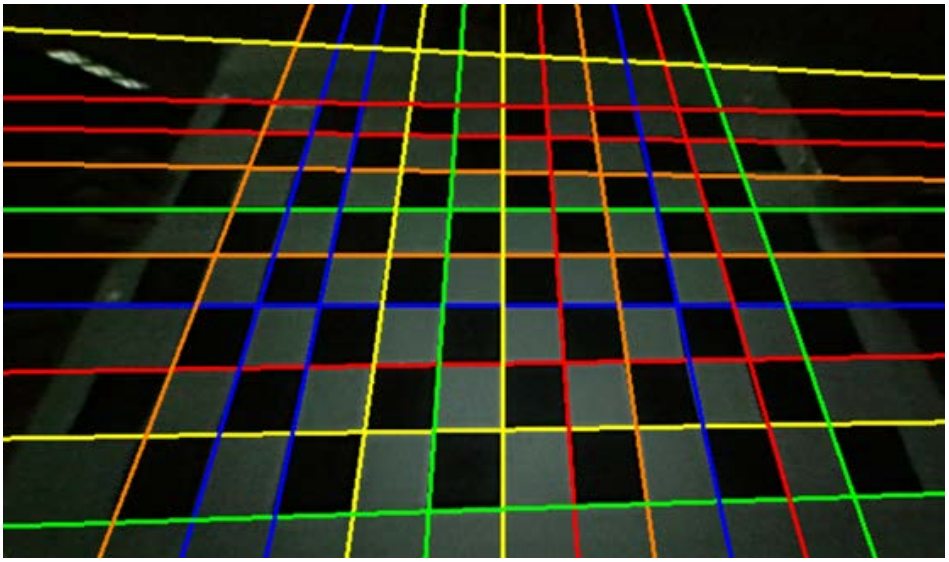
4. Ảnh Chessboard_0481

Kết quả:

```
The chessboard size is 9x12 (9 rows and 12 columns)
Accuracy: 93.75%
```

Nguyên nhân: Tim thừa 1 đường ngang ở cạnh trên của bàn cờ

Đường thẳng tìm được trên ảnh:



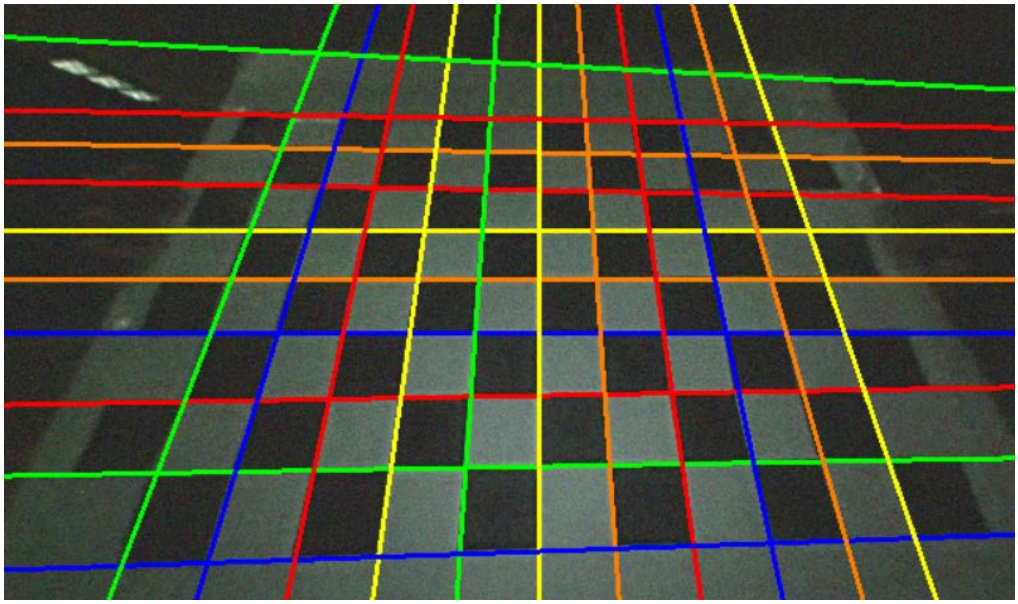
5. Ảnh Chessboard_0481_0

Kết quả:

```
The chessboard size is 9x12 (9 rows and 12 columns)
Accuracy: 93.75%
```

Nguyên nhân: Tim thừa 1 đường ngang ở cạnh trên của bàn cờ.

Đường thẳng tìm được trên ảnh:



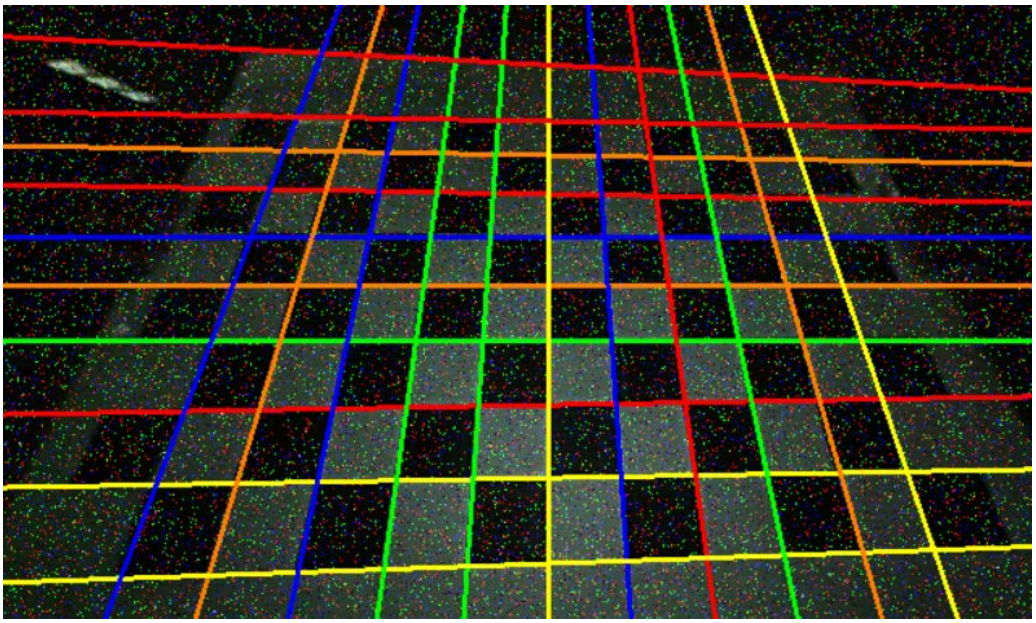
6. Ảnh Chessboard_0481_1

Kết quả:

```
The chessboard size is 9x12 (9 rows and 12 columns)
Accuracy: 93.75%
```

Nguyên nhân: Tim thừa 1 đường ngang ở cạnh trên của bàn cờ.

Đường thẳng tìm được trên ảnh:



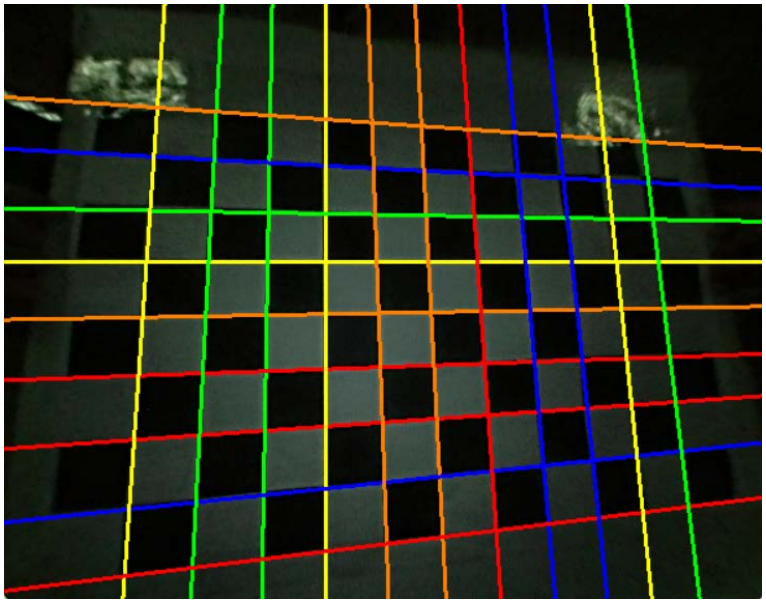
7. Ảnh Chessboard_0511

Kết quả:

```
The chessboard size is 8x12 (8 rows and 12 columns)
Accuracy: 100.0%
```

Nguyên nhân: Tìm được tất cả các đường thẳng hợp lệ.

Đường thẳng tìm được trên ảnh:



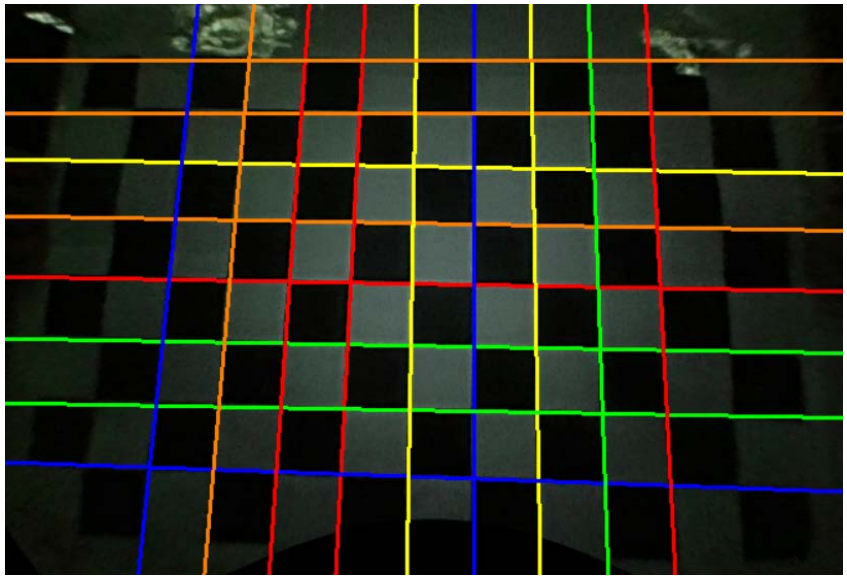
8. Ảnh Chessboard_0541

Kết quả:

```
The chessboard size is 7x10 (7 rows and 10 columns)
Accuracy: 85.41666666666667%
```

Nguyên nhân: Không tìm được 2 đường dọc ở sát trái và sát phải của bàn cờ, cũng như 1 đường ngang ở viền dưới của bài cờ (Do có vùng đen ở cạnh dưới của ảnh).

Đường thẳng tìm được trên ảnh:



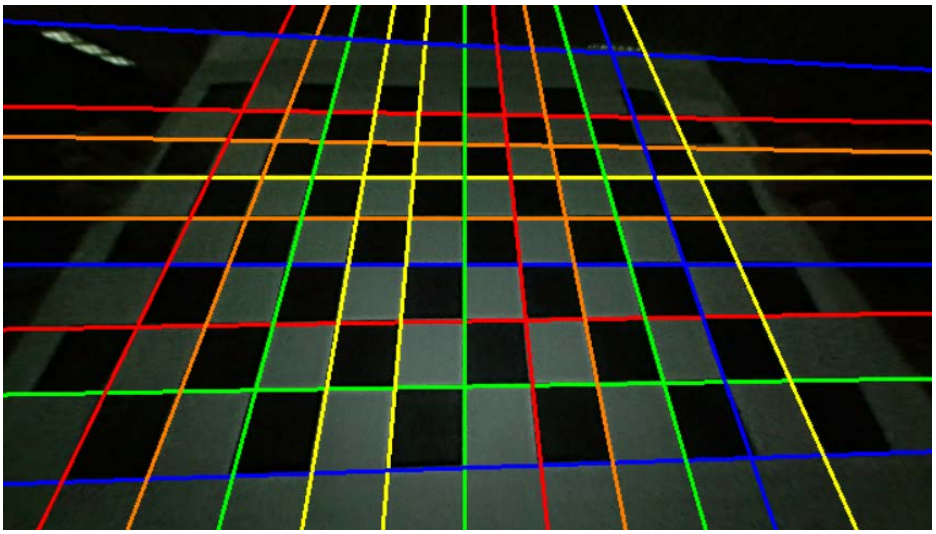
9. Ảnh Chessboard_0601

Kết quả:

```
The chessboard size is 8x12 (8 rows and 12 columns)
Accuracy: 100.0%
```

Nguyên nhân: Không tìm được đường ngang ở viền trên của bàn cờ, nhưng tìm thừa 1 đường ngang ở cạnh trên của bàn cờ.

Đường thẳng tìm được trên ảnh:



10. Ảnh Chessboard_0631

Kết quả:

```
The chessboard size is 7x12 (7 rows and 12 columns)
Accuracy: 93.75%
```

Nguyên nhân: Không tìm được 1 đường ngang ở viền dưới của bài cờ (Do có vùng đen ở cạnh dưới của ảnh).

Đường thẳng tìm được trên ảnh: