

컴퓨터비전 과제 1

#1. Checkerboard

References

1. 강의 교안
2. <https://github.com/sunkyoo/opencv4cvml>
3. <https://ks-jun.tistory.com/m/198> → 투시변환 꼭짓점 지정 및 투시변환 매커니즘 참조
4. <https://nowprogramming.tistory.com/18> → 접근방식 참조
5. ChatGPT, Claude

풀이 과정

임의의 체커 보드판 이미지를 고려해야 하기 때문에, 임의의 다양한 각도에서 촬영된 체커보드판을 투시변환을 이용하여 먼저 2차원 평면으로 변환시킨 다음 처리해야겠다고 생각했습니다.

따라서 먼저

`order_points()`, `four_point_transform()` 메소드 그리고 `findContours()` 메소드를 활용해서 체커보드판의 윤곽선을 찾아내고(max 메소드로 가장 큰 윤곽선을 검출 = 체커보드판의 윤곽선일 것이라 가정함)

윤곽선에서 4개의 꼭짓점을 추출한 뒤

4개의 꼭짓점 정보로부터 투시변환을 수행하였습니다. 이 영상을

`warped` 변수에 저장했습니다.(위 과정은 #3에서 좀 더 상세히 설명할 예정)

그 다음 2차원 평면상에 놓인 체커보드에서 각 칸의 개수를 세기 위해서 우선 그레이스케일로 변환을 수행하였고 이를 `gray` 변수에 저장하였습니다.

`gray` 영상을 바탕으로 체커보드의 각 칸의 개수를 세는 `detect_squares()` 메소드를 적용하였습니다.

- `detect_squares()` 메소드

- 2차원 평면으로 변환된 체커보드에서 각 칸의 윤곽선을 검출하는 메소드입니다. 입력 이미지를 그레이스케일로 받은 후 `GaussianBlur()`로 노이즈를 줄인 뒤 Canny 에지 검출기를 사용하여 윤곽선을 찾습니다.
- 에지를 `dilate()`로 팽창 연산하여 윤곽선을 더 뚜렷하게 만들고 색 반전을 적용합니다.
- `findContours()`로 외곽선을 검출한 후, 각 윤곽선이 사각형인지 확인하고 후보 사각형으로 선택했습니다.
- 선택된 후보 사각형들의 면적을 각각 계산한 값을 `area` 리스트 변수에 저장하였습니다. 이 리스트로 사각형으로 판정된 윤곽선 면적의 평균과 표준편차를 계산하여, 평균치와 차이가 적은 윤곽선만 추가하는 검증을 통해 체커보드 칸의 진위성이 검증된 체커보드 칸만 인식할 수 있도록 매커니즘을 설계했습니다.

- `determine_board_size()` 메소드

- **유의사항 :** 임의의 체커 보드판 이미지는 임의의 각도, 임의의 판 색상, 임의의 밝기, 판에 말(동그라미)이 올려져 있는지의 여부 등에 따라 매우 상이하기 때문에 모든 조건을 필터링하여 정확하게 모든 칸의 수를 판별하는 것은 제 능력으로는 한계가 있습니다.

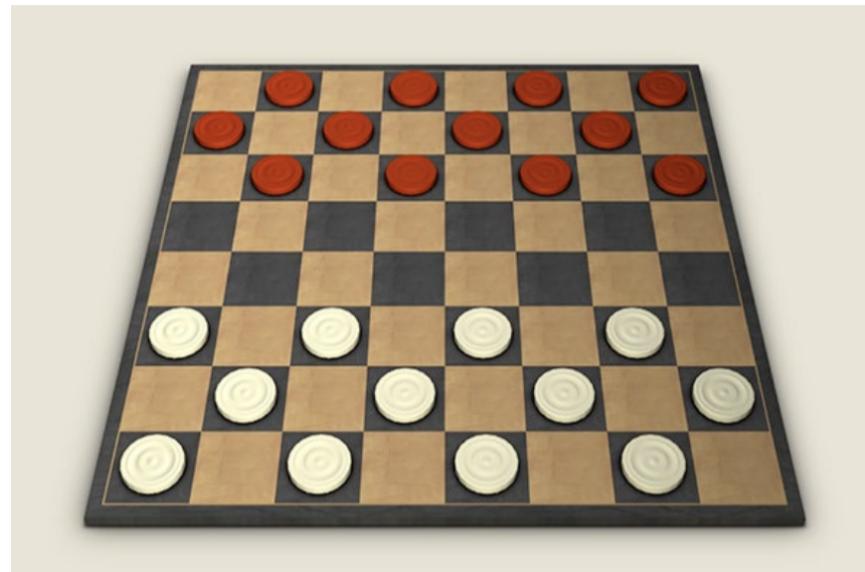
따라서 체커보드 크기 추정 시, 개수에 따른 사이즈 판별 상한/하한을 널널하게 설정한 점 양해 부탁드리겠습니다.

- 검출된 사각형 수를 기준으로 체커보드의 크기를 추정하는 메소드입니다.
- 사각형 개수가 10개 ~ 64개 사이면 "8 x 8",
65개 ~ 100개 사이면 "10 x 10"으로 판별하며, 그 외는 불가능한 이미지로 처리하게 설계했습니다.

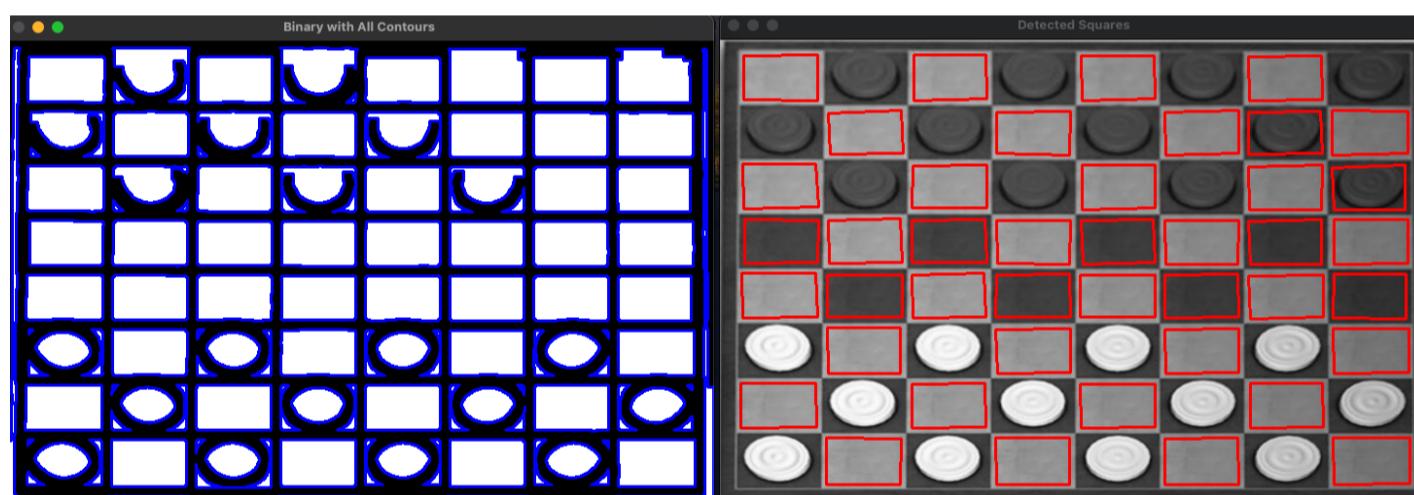
더욱 디테일한 요소들은 소스코드 내 주석 상에 적어두었습니다.

Example Images

1. 원본 이미지



2. 실행결과 이미지



3. 판별 결과 콘솔

```
○ ikjoon@IJui-MacBookAir hw1 % python3 hw1_1.py 1.png
검출된 사각형 개수: 42
추정되는 체커보드 크기: 8 x 8
```

#2. Perspective Transform (수동)

References

1. 강의 교안
2. <https://github.com/sunkyoo/opencv4cvml>

풀이 과정

8장 강의 교안에서, 투시 변환을 적용하여 사용자가 마우스로 카드 모서리 좌표 4개를 좌상단부터 시계방향으로 선택하면, 해당 카드를 반듯한 직사각형 형태로 투시 변환하여 화면에 출력하는 예제 소스코드를 그대로 인용하였습니다.

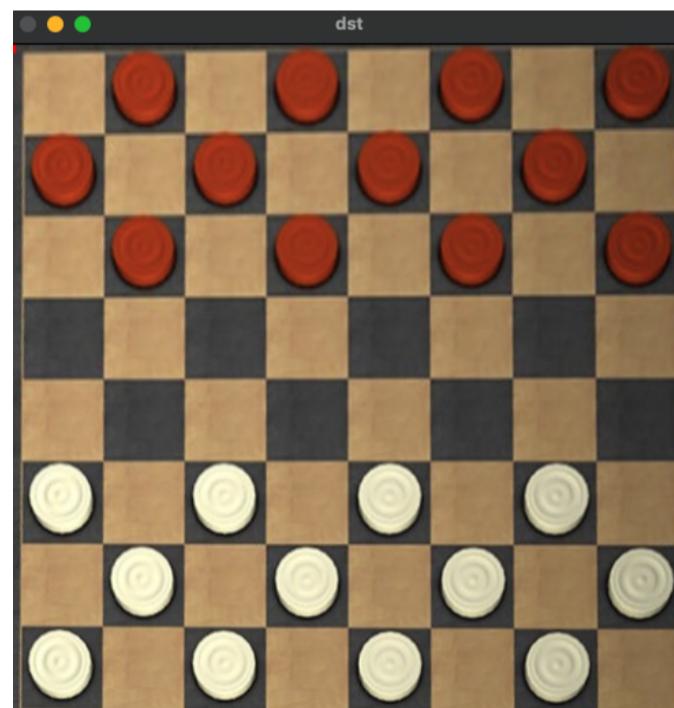
이때, 출력 사이즈만 체커보드가 정사각형인지를 고려하여 `w = 500, h = 500` 으로 설정하였습니다.

Example Images

- 원본 이미지에서 4개의 꼭짓점을 시계방향으로 선택한 src 이미지



- 투시변환 결과 dst 이미지



#3. Perspective Transform (자동)

References

- 강의 교안
- <https://github.com/sunkyoo/opencv4cvml>
- <https://ks-jun.tistory.com/m/198> → 투시변환 꼭짓점 지정 및 투시변환 매커니즘 참조

풀이 과정

앞서 #2에서 진행한 과정을 자동으로 수행하는 매커니즘을 구현해야 했습니다. 처음 과제를 받고 #1의 풀이과정을 생각했을 때 임의의 각도에서 촬영된 임의의 이미지를 먼저 2차원 평면으로 옮겨서 처리해야겠다고 생각해서, #1보다 #3을 먼저 해결하게 되었습니다. 메소드 및 매커니즘 설명은 아래와 같습니다.

1. `order_points()`

- 4개의 꼭짓점 좌표를 받아 사각형으로 정의할 수 있도록 좌표를 정렬합니다.
- 좌표 합의 최소, 최대값을 이용해 좌측 상단과 우측 하단 좌표를 찾고, 좌표 차의 최소값과 최대값으로 우측 상단과 좌측 하단 좌표를 찾습니다. 투시 변환을 위한 기준 좌표로 사용됩니다.

2. `four_point_transform()`

- 주어진 이미지에서 체커보드의 윤곽선을 투시 변환하여 평면화된 정사각형 형태로 변환합니다.
- `order_points()`로 좌표를 정렬한 후 좌측 상단, 우측 상단, 우측 하단, 좌측 하단의 좌표(tl, tr, br, bl)를 각각 지정합니다.
- 각 변의 길이인 widthA, widthB, heightA, heightB를 계산하여 변환할 이미지의 최대 폭(maxWidth)과 높이(maxHeight)를 결정합니다.
- `dst`를 정의하고 `getPerspectiveTransform()`을 사용하여 투시 변환 행렬(M)을 계산한 후, `warpPerspective()`로 이미지에 실제 변환을 적용합니다.

3. `main()`

- 원본 이미지를 불러와서 `cv2.Canny()`로 캐니 에지 검출을 수행하고, `cv2.dilate()`로 팽창 연산을 수행하여 윤곽선을 더욱 뚜렷하게 만듭니다.
- 체커보드의 외곽선을 찾기 위해 모든 윤곽선 중에서 가장 큰 윤곽선을 선택합니다. `findContours()`로 모든 외곽선을 추출한 후, `max(contours, key = cv2.contourArea)`로 가장 큰 윤곽선을 선택합니다.
- `arcLength()`로 윤곽선의 둘레를 구하고, 이 값의 2%를 epsilon으로 설정하여 `approxPolyDP()`로 윤곽선을 단순화합니다. 꼭짓점이 4개인 경우 체커보드의 외곽 윤곽선으로 판별하고 투시 변환을 수행합니다.
- `four_point_transform()` 메소드를 호출해 투시 변환을 수행하여 체커보드를 평면화된 이미지로 변환합니다. 투시 변환된 이미지를 정해진 크기 (w = 500, h = 500)으로 조정하여 `warped` 변수에 저장합니다.

Example Images

1. #2와 동일한 이미지의 자동 투시 변환 결과



original image

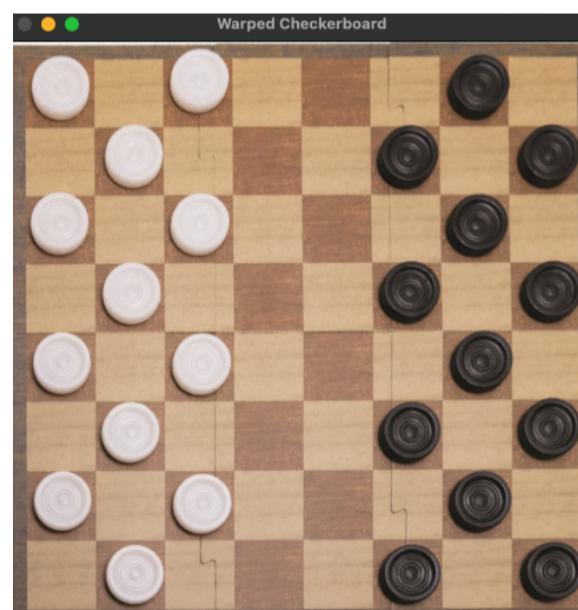


warped checkerboard

2. 상이한 이미지의 자동 투시 변환 결과



original image



warped checkerboard

#4. Counting the pieces

References

1. 강의 교안
2. <http://www.gisdeveloper.co.kr/?p=6726> → 접근방식 참조
3. <https://stackoverflow.com/questions/58109962/how-to-optimize-circle-detection-with-python-opencv> → 파라미터 참조
4. ChatGPT, Claude

풀이 과정

- **유의사항 :** 임의의 체커 보드판 이미지는 임의의 각도, 임의의 판 색상, 임의의 밝기, 판에 말(동그라미)이 올려져 있는지의 여부 등에 따라 매우 상이하기 때문에 모든 조건을 필터링하여 정확하게 밝은 색 말과 어두운 색 말의 수를 판별하는 것은 제 능력으로는 한계가 있었습니다.
따라서 4번 같은 경우에는 일반화(범용)를 위해 소스코드를 최대한 단순화하고자 했습니다.

우선 원본 컬러 이미지를 `cvtColor()` 메소드로 그레이스케일로 변환하였습니다. 이미지의 노이즈를 줄이기 위해 `GaussianBlur()` 메소드로 가우시안 블러링을 수행하여, 원 검출 시 노이즈를 최소화하고자 하였습니다.

그 다음에는 가장 중요한 원 검출 메소드, `HoughCircles()` 을 이용하였습니다. 체커 말을 원으로 인식하기 위해 허프 변환을 사용하여 원형 객체를 검출합니다. 파라미터 설정은 상기한 블로그 예제를 참조하였고, 휴리스틱하게 임의의 이미지들에 대해 설정해가며 조절했습니다. 최종 파라미터 설정값은 아래와 같습니다.

- `dp = 1` : 해상도를 원본과 동일하게 설정하여 원을 정확히 감지합니다.
- `minDist = 20` : 검출된 원들 간의 최소 거리로, 말이 너무 가까이 있는 경우 중복 검출을 방지합니다.
- `param1 = 50, param2 = 30` : 가장자리 검출 민감도와 원 검출 임계값으로, 이를 조정하여 정확도를 유의미한 수준으로 끌어올렸습니다.
- `minRadius = 10, maxRadius = 50` : 검출할 원의 반지름 범위를 제한하여 체커 말 크기에 적절하게 설정합니다.

검출된 원이 있는 경우 각 원의 중심점을 기준으로 `gray[y, x]` (밝기)값을 가져와 밝은 색과 어두운 색을 구분합니다.

밝기 임계값(brightness)을 128로 설정하여, 밝기 값이 128 이상인 경우를 흰색으로, 그렇지 않은 경우를 검은색으로 간주합니다. 분류된 결과는 각각 `white_count` 와 `black_count` 에 누적되고 최종적으로 출력하게 설계했습니다.

Example Images

1. 말이 올려져 있는 체커보드 원본 이미지 및 판별 결과 콘솔(오차 발생 예시)



```
● ikjoon@IJui-MacBookAir hw1 % python3 hw1_4.py 1.png
  w:13 b:12
```

2. 상이한 사진 및 판별 결과 콘솔(정상 작동 예시)



```
● ikjoon@IJui-MacBookAir hw1 % python3 hw1_4.py 2.png
  w:12 b:12
```

감사합니다.