컴퓨터 비전 HW1

숭실대학교 컴퓨터학부 20202920 조민혁

소스코드와 설명

1.hw\_1

import cv2  
import numpy as np  
import sys  
  
def count\_checkerboard\_squares(image\_path):  
 image = cv2.imread(image\_path) # 이미지 파일 읽기  
  
 if image is None:  
 print("이미지를 불러올 수 없습니다.")  
 return  
  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 # 대비 향상을 위한 CLAHE 적용  
 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))  
 gray = clahe.apply(gray)  
 # 가우시안 블러로 노이즈 감소  
 blurred = cv2.GaussianBlur(gray, (3, 3), 0)  
 # 에지 검출  
 edges = cv2.Canny(blurred, 50, 150, apertureSize=3)  
  
 # 허프 변환을 이용한 선 검출  
 lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=100,  
 minLineLength=50, maxLineGap=10)  
  
 if lines is None:  
 print("선을 검출할 수 없습니다.")  
 return  
  
 # 수직선과 수평선 분류  
 vertical\_lines = []  
 horizontal\_lines = []  
  
 for line in lines:  
 for x1, y1, x2, y2 in line:  
 # 기울기 계산  
 if x2 - x1 == 0:  
 angle = 90  
 else:  
 angle = np.degrees(np.arctan2(y2 - y1, x2 - x1))  
  
 # 수직선 (기울기 약 90도)  
 if abs(angle) > 80:  
 vertical\_lines.append((x1, y1, x2, y2))  
 # 수평선 (기울기 약 0도)  
 elif abs(angle) < 10:  
 horizontal\_lines.append((x1, y1, x2, y2))  
  
 if not vertical\_lines or not horizontal\_lines:  
 print("충분한 수직선 또는 수평선을 검출하지 못했습니다.")  
 return  
  
 # 수직선과 수평선 정렬  
 vertical\_lines.sort(key=lambda x: x[0])  
 horizontal\_lines.sort(key=lambda x: x[1])  
  
 # 수직선과 수평선의 x, y 좌표 추출  
 vertical\_positions = [ (x1 + x2) // 2 for x1, y1, x2, y2 in vertical\_lines ]  
 horizontal\_positions = [ (y1 + y2) // 2 for x1, y1, x2, y2 in horizontal\_lines ]  
  
 # 중복되는 좌표 제거 (근사값 처리)  
 def remove\_duplicates(positions, tolerance=10):  
 if not positions:  
 return []  
 positions = sorted(positions)  
 unique\_positions = [positions[0]]  
 for pos in positions[1:]:  
 if abs(pos - unique\_positions[-1]) > tolerance:  
 unique\_positions.append(pos)  
 return unique\_positions  
  
 unique\_vertical\_positions = remove\_duplicates(vertical\_positions)  
 unique\_horizontal\_positions = remove\_duplicates(horizontal\_positions)  
  
 num\_cols\_detected = len(unique\_vertical\_positions) - 1  
 num\_rows\_detected = len(unique\_horizontal\_positions) - 1  
  
 # 예상되는 체커보드의 크기 (8x8 또는 10x10)  
 def decide\_size(num):  
 return 8 if abs(num - 8) < abs(num - 10) else 10  
  
 num\_cols = decide\_size(num\_cols\_detected)  
 num\_rows = decide\_size(num\_rows\_detected)  
  
 # 최종적으로 결정된 행과 열의 수를 출력  
 if num\_rows == num\_cols and num\_rows in [8, 10]:  
 print(f"{num\_rows} x {num\_cols}")  
 else:  
 # 행과 열이 다르게 검출되었거나 8, 10이 아닌 경우 가까운 값으로 조정  
 avg\_num = int(round((num\_rows\_detected + num\_cols\_detected) / 2))  
 final\_size = decide\_size(avg\_num)  
 print(f"{final\_size} x {final\_size}")  
  
 # 결과 시각화 (옵션)  
 # 수직선과 수평선 그리기  
 for x in unique\_vertical\_positions:  
 cv2.line(image, (x, 0), (x, image.shape[0]), (0, 255, 0), 1)  
 for y in unique\_horizontal\_positions:  
 cv2.line(image, (0, y), (image.shape[1], y), (255, 0, 0), 1)  
  
 # 교차점 그리기  
 for x in unique\_vertical\_positions:  
 for y in unique\_horizontal\_positions:  
 cv2.circle(image, (x, y), 5, (0, 0, 255), -1)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) != 2:  
 print("사용법: python hw1\_1.py <이미지 경로>")  
 else:  
 count\_checkerboard\_squares(sys.argv[1])

설명 :

count\_checkerboard\_squares는 입력된 이미지에서 체커보드 패턴을 감지하고, 체커보드의 행과 열 수를 계산하여 출력합니다.

구성 요소:

이미지 로드 및 그레이스케일 변환:

입력 이미지 파일을 cv2.imread로 읽고, cv2.cvtColor를 사용해 그레이스케일로 변환합니다.

CLAHE로 대비 향상:

cv2.createCLAHE로 이미지의 대비를 향상하여 더 잘 보이도록 조정하고, 균일하게 처리될 수 있도록 대비 한계를 적용합니다.

가우시안 블러 및 에지 검출:

cv2.GaussianBlur를 사용해 노이즈를 줄인 후 cv2.Canny 에지 검출기로 이미지에서 에지(윤곽선)를 찾습니다.

허프 변환을 통한 선 검출:

cv2.HoughLinesP를 사용하여 이미지에서 선을 검출합니다. 이 함수는 선의 최소 길이와 허용 간격을 기준으로 선을 검출합니다.

검출된 선이 없으면 "선을 검출할 수 없습니다"라는 메시지를 출력하고 종료합니다.

기울기에 따른 수직선과 수평선 분류:

각 선의 기울기를 계산하여 기울기가 90도에 가까운 선은 수직선, 0도에 가까운 선은 수평선으로 분류합니다.

수직선과 수평선이 충분히 검출되지 않으면 "충분한 수직선 또는 수평선을 검출하지 못했습니다"라는 메시지를 출력합니다.

좌표 정렬 및 중복 좌표 제거:

수직선과 수평선을 각각 정렬하고, 좌표가 근사값으로 중복된 경우를 제거하여 고유한 좌표를 남깁니다.

remove\_duplicates 함수는 두 좌표 사이의 간격이 일정 임계값 이상이면 고유한 좌표로 간주하여 중복을 제거합니다.

행과 열 계산 및 체커보드 크기 결정:

고유한 수직선과 수평선의 개수를 기반으로, 각각 열과 행의 수를 계산합니다.

계산된 행과 열의 수가 체커보드의 일반적인 크기(8x8 또는 10x10)와 가까운지 확인하여 최종 행과 열 수를 결정합니다.

예상되는 행/열이 같으면 해당 값을 출력하고, 그렇지 않으면 평균값을 기준으로 조정하여 가까운 크기(8 또는 10)로 맞춥니다.

2. 보조 함수 설명

remove\_duplicates는 선 좌표 리스트에서 중복된 값을 제거하여 고유한 좌표만 남기는 함수입니다.

작동 원리:

각 좌표를 정렬하고, 인접한 좌표 간의 차이가 임계값(tolerance) 이상인 경우만 고유 좌표로 간주하여 중복을 제거합니다.

decide\_size

역할: 계산된 행 또는 열의 수가 체커보드의 일반적인 크기(8x8 또는 10x10) 중 어느 것에 가까운지 판별합니다.

결과 :

해당 이미지는 Image.png로 결과는 아래와 같습니다.

체커, 실내 게임 및 스포츠, 보드게임, 게임이(가) 표시된 사진

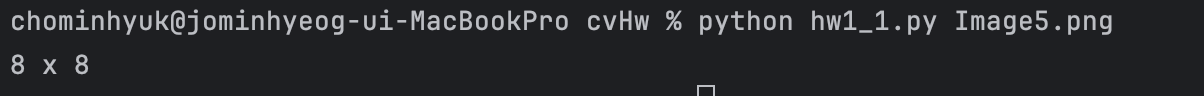
자동 생성된 설명



해당 이미지는 Image5.png이고 결과는 아래와 같습니다.

체커, 실내 게임 및 스포츠, 보드게임, 테이블 게임이(가) 표시된 사진

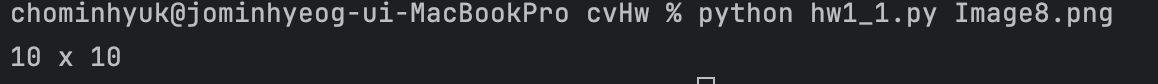
자동 생성된 설명



해당 이미지는 Image8.png이고 결과는 아래와 같습니다.

체커, 보드게임, 실내 게임 및 스포츠, 테이블 게임이(가) 표시된 사진

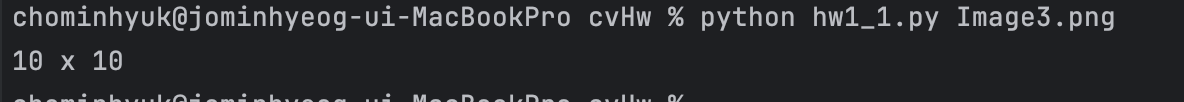
자동 생성된 설명



해당 이미지는 Image3.png이고 결과는 아래와 같습니다.

사각형, 체스, 직사각형, 보드게임이(가) 표시된 사진

자동 생성된 설명



2.hw\_2

import cv2 # OpenCV 라이브러리 import  
import numpy as np # Numpy 라이브러리 import  
import sys # 시스템 관련 모듈 import  
  
# 전역 변수 설정  
points = [] # 사용자가 선택한 4개의 좌표를 저장할 리스트  
image = None # 현재 이미지  
original\_image = None # 원본 이미지  
  
# 마우스 클릭 이벤트 처리 함수  
def on\_mouse\_click(event, x, y, flags, param):  
 global points, image  
 # 왼쪽 버튼 클릭 시, 최대 4개의 좌표를 저장  
 if event == cv2.EVENT\_LBUTTONDOWN and len(points) < 4:  
 points.append([x, y]) # 클릭한 좌표 추가  
 cv2.circle(image, (x, y), 5, (0, 255, 0), -1) # 클릭한 위치에 초록색 원 그리기  
 cv2.imshow("Image", image) # 이미지 갱신하여 표시  
  
# 원근 변환 함수  
def perspective\_transform(image\_path):  
 global image, original\_image, points  
 image = cv2.imread(image\_path) # 이미지 파일 읽기  
  
 if image is None:  
 print("이미지를 불러올 수 없습니다.")  
 return  
  
 original\_image = image.copy() # 원본 이미지 복사본 생성  
  
 cv2.namedWindow("Image") # 이미지 창 생성  
 cv2.imshow("Image", image) # 이미지를 화면에 표시  
 cv2.setMouseCallback("Image", on\_mouse\_click) # 마우스 클릭 이벤트 등록  
  
 # 사용자가 4개의 점을 선택할 때까지 기다림  
 while True:  
 key = cv2.waitKey(1) & 0xFF  
 if len(points) == 4: # 4개의 점이 선택되면 루프 종료  
 break  
 elif key == 27: # ESC 키를 누르면 프로그램 종료  
 print("프로그램을 종료합니다.")  
 cv2.destroyAllWindows()  
 return # sys.exit() 대신 함수 종료로 변경  
  
 cv2.destroyWindow("Image") # 이미지 창 닫기  
  
 pts1 = np.float32(points) # 선택한 좌표를 float32 타입으로 변환  
  
 # 변환 후 이미지의 폭과 높이 계산  
 width\_top = np.linalg.norm(pts1[0] - pts1[1]) # 상단 폭  
 width\_bottom = np.linalg.norm(pts1[3] - pts1[2]) # 하단 폭  
 width = max(int(width\_top), int(width\_bottom)) # 폭 중 큰 값 선택  
  
 height\_left = np.linalg.norm(pts1[0] - pts1[3]) # 왼쪽 높이  
 height\_right = np.linalg.norm(pts1[1] - pts1[2]) # 오른쪽 높이  
 height = max(int(height\_left), int(height\_right)) # 높이 중 큰 값 선택  
  
 # 원근 변환 후 목적지 좌표 설정  
 pts2 = np.float32([[0, 0], [width, 0], [width, height], [0, height]])  
  
 # 원근 변환 행렬 계산  
 matrix = cv2.getPerspectiveTransform(pts1, pts2)  
 result = cv2.warpPerspective(original\_image, matrix, (width, height)) # 변환 적용  
  
 cv2.imshow("Transformed", result) # 변환된 이미지 표시  
  
 # 임의의 키 입력 시 종료  
 cv2.waitKey(0) # 키 입력 대기  
 cv2.destroyAllWindows() # 모든 창 닫기  
  
# 메인 함수  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) != 2:  
 print("사용법: python hw1\_2.py <이미지 경로>")  
 else:  
 perspective\_transform(sys.argv[1])

설명 :

1. 전역 변수 및 라이브러리

points: 원근 변환에 사용할 네 개의 좌표를 저장하는 리스트입니다.

image와 original\_image: 현재 작업 중인 이미지와 원본 이미지를 저장합니다.

cv2, np, sys: OpenCV는 이미지 처리, Numpy는 배열 연산, sys는 시스템 관련 작업을 위한 라이브러리입니다.

2. on\_mouse\_click 함수

이 함수는 사용자가 마우스를 클릭해 선택한 좌표를 이미지에 표시합니다.

cv2.EVENT\_LBUTTONDOWN 이벤트를 통해 왼쪽 버튼이 클릭될 때 좌표를 points 리스트에 저장하고, 해당 위치에 초록색 원을 표시하여 시각적으로 피드백을 제공합니다.

3. perspective\_transform 함수

주어진 이미지 파일을 불러와 사용자가 선택한 네 개의 점을 기준으로 원근 변환을 수행합니다.

1) 이미지 로딩 및 창 설정

cv2.imread를 사용하여 입력 경로에서 이미지를 읽고, original\_image에 복사본을 저장합니다.

cv2.imshow로 이미지를 창에 띄우고, cv2.setMouseCallback으로 on\_mouse\_click 이벤트를 연결합니다.

2) 좌표 선택 및 대기

4개의 좌표를 선택할 때까지 무한 루프를 유지하고, 각 클릭으로 좌표를 추가합니다.

4개의 좌표가 입력되면 루프를 종료하고 원근 변환을 시작합니다.

3) 원근 변환 수행

사용자가 선택한 네 좌표(pts1)를 목적지 좌표(pts2)로 이동시키는 변환을 계산합니다.

목적지 좌표(pts2)는 사각형 형태로, 변환 후 폭과 높이를 계산하여 설정합니다.

cv2.getPerspectiveTransform를 통해 변환 행렬을 계산하고, cv2.warpPerspective를 사용해 변환을 적용합니다.

4) 결과 표시

변환된 이미지를 새 창에 표시하고, ESC 또는 q 키 입력을 대기하여 프로그램을 종료합니다.

4. \_\_main\_\_ 메인 함수

프로그램을 명령줄에서 실행할 때 이미지 경로를 인자로 받으며, 경로가 없거나 잘못되면 사용법을 안내합니다.

결과 :

사각형, 실내 게임 및 스포츠, 보드게임, 스크린샷이(가) 표시된 사진

자동 생성된 설명

실내 게임 및 스포츠, 체커, 게임, 보드게임이(가) 표시된 사진

자동 생성된 설명

제출했던 이미지 전부에서 실행되는 것을 확인해볼 수 있었습니다.

3.hw\_3

import cv2  
import numpy as np  
import sys  
  
  
def automatic\_perspective\_transform(image\_path):  
 image = cv2.imread(image\_path)  
  
 if image is None:  
 print("이미지를 불러올 수 없습니다.")  
 return  
  
 # 이미지를 그레이스케일로 변환  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 # 가우시안 블러 적용  
 gray\_blurred = cv2.GaussianBlur(gray, (5, 5), 0)  
 # 적응형 이진화  
 thresh = cv2.adaptiveThreshold(gray\_blurred, 255,  
 cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,  
 cv2.THRESH\_BINARY\_INV, 11, 2)  
 # 엣지 검출  
 edges = cv2.Canny(thresh, 50, 150)  
 # 닫힘 연산  
 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))  
 closed = cv2.morphologyEx(edges, cv2.MORPH\_CLOSE, kernel)  
 # 컨투어 찾기  
 contours, \_ = cv2.findContours(closed, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 if not contours:  
 print("윤곽을 찾을 수 없습니다.")  
 return  
  
 # 모든 컨투어를 면적 기준으로 정렬  
 contours = sorted(contours, key=cv2.contourArea, reverse=True)  
  
 # 사각형 찾기  
 for contour in contours:  
 epsilon = 0.02 \* cv2.arcLength(contour, True)  
 approx = cv2.approxPolyDP(contour, epsilon, True)  
  
 if len(approx) == 4:  
 pts = approx.reshape(4, 2)  
 rect = order\_points(pts)  
 break  
 else:  
 print("사각형을 찾을 수 없습니다.")  
 return  
  
 # 투시 변환 적용  
 maxWidth, maxHeight = get\_max\_width\_height(rect)  
 dst = np.array([  
 [0, 0],  
 [maxWidth - 1, 0],  
 [maxWidth - 1, maxHeight - 1],  
 [0, maxHeight - 1]], dtype="float32")  
  
 matrix = cv2.getPerspectiveTransform(rect, dst)  
 result = cv2.warpPerspective(image, matrix, (int(maxWidth), int(maxHeight)))  
  
 cv2.imshow("Transformed", result)  
  
 cv2.waitKey(0) # 키 입력 대기  
 cv2.destroyAllWindows()  
 sys.exit()  
  
  
def order\_points(pts):  
 rect = np.zeros((4, 2), dtype="float32")  
 s = pts.sum(axis=1)  
 diff = np.diff(pts, axis=1)  
  
 rect[0] = pts[np.argmin(s)] # 좌상단  
 rect[2] = pts[np.argmax(s)] # 우하단  
 rect[1] = pts[np.argmin(diff)] # 우상단  
 rect[3] = pts[np.argmax(diff)] # 좌하단  
  
 return rect  
  
  
def get\_max\_width\_height(rect):  
 (tl, tr, br, bl) = rect  
  
 widthA = np.linalg.norm(br - bl)  
 widthB = np.linalg.norm(tr - tl)  
 maxWidth = max(int(widthA), int(widthB))  
  
 heightA = np.linalg.norm(tr - br)  
 heightB = np.linalg.norm(tl - bl)  
 maxHeight = max(int(heightA), int(heightB))  
  
 return (maxWidth, maxHeight)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) != 2:  
 print("사용법: python hw1\_3.py <이미지 경로>")  
 else:  
 automatic\_perspective\_transform(sys.argv[1])

설명 :

hw\_3은 hw\_2에 진행했던 과정을 사용자가 점을 찍어 입력하는 것이 아닌 자동으로 판별하여 체스판 이미지를 추출하는 것입니다.

automatic\_perspective\_transform 함수 설명

주어진 이미지에서 사각형 모양의 윤곽선을 자동으로 탐지하고, 해당 영역에 대해 원근 변환을 수행하는 함수입니다.

함수의 구성 요소:

이미지 로드 및 그레이스케일 변환:

cv2.imread를 사용하여 이미지 파일을 로드하고, cv2.cvtColor로 그레이스케일 이미지로 변환합니다.

가우시안 블러 및 이진화:

cv2.GaussianBlur로 그레이스케일 이미지에 블러를 적용해 노이즈를 줄이고, cv2.adaptiveThreshold로 이진화를 수행하여 윤곽이 명확한 이미지를 생성합니다.

엣지 검출 및 닫힘 연산:

cv2.Canny를 통해 엣지를 검출하고, cv2.morphologyEx의 닫힘 연산으로 간격을 메워 윤곽을 더욱 선명하게 만듭니다.

컨투어 탐지 및 정렬:

cv2.findContours를 이용해 이미지에서 외곽선을 탐지하고, 면적이 큰 순서로 정렬하여 가장 큰 사각형을 찾는 데 집중합니다.

사각형 검출:

cv2.approxPolyDP로 단순화된 윤곽을 만들어 네 개의 꼭짓점을 찾고, 사각형 모양인지 확인합니다.

사각형이 발견되면 좌표를 order\_points 함수에 전달해 순서대로 정렬합니다.

투시 변환 및 결과 표시:

get\_max\_width\_height 함수를 통해 최종적으로 변환된 이미지의 폭과 높이를 계산하고, cv2.getPerspectiveTransform과 cv2.warpPerspective로 변환을 적용해 결과 이미지를 생성합니다.

최종 이미지를 화면에 출력하고, 사용자가 ESC 또는 'q' 키를 누를 때까지 대기합니다.

order\_points은 감지된 사각형의 네 개 좌표를 좌상단, 우상단, 우하단, 좌하단 순으로 정렬합니다.

결과:

실내 게임 및 스포츠, 게임, 보드게임, 체커이(가) 표시된 사진

자동 생성된 설명

실내 게임 및 스포츠, 게임, 체커, 보드게임이(가) 표시된 사진

자동 생성된 설명

hw\_2와 마찬가지로 경계가 명확한 체스판에 대해서는 전부 수행 가능함을 확인하였습니다. 사용자가 이미지 경로를 설정하기만 하면 해당 이미지에 대해서 동작합니다.

4.hw\_4

import cv2  
import numpy as np  
import sys  
  
def get\_chess(image\_path):  
 # 이미지 읽기  
 image = cv2.imread(image\_path)  
 if image is None:  
 print("이미지를 불러올 수 없습니다. 경로를 확인하세요.")  
 return 0, 0 # 두 개의 값 반환으로 수정하여 예외 발생 방지  
  
 # 자동 투시 변환  
 image = automatic\_perspective\_transform(image)  
 if image is None:  
 print("투시 변환에 실패했습니다.")  
 return 0, 0 # 두 개의 값 반환으로 수정하여 예외 발생 방지  
  
 # 흑백 이미지로 변환 및 블러 처리  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 gray = cv2.medianBlur(gray, 5)  
  
 # 원 검출  
 circles = cv2.HoughCircles(  
 gray,  
 cv2.HOUGH\_GRADIENT,  
 dp=1,  
 minDist=12, # 원 간 최소 거리  
 param1=65, # 캐니 에지 검출기 상위 임계값  
 param2=19, # 원 검출 임계값  
 minRadius=9, # 검출할 원의 최소 반지름  
 maxRadius=30 # 검출할 원의 최대 반지름  
 )  
  
 if circles is not None:  
 circles = np.int32(np.around(circles[0, :]))  
  
 # 반지름을 기준으로 원을 내림차순 정렬 후 중복 제거  
 circles = sorted(circles, key=lambda x: x[2], reverse=True)  
 filtered\_circles = []  
  
 for c in circles:  
 center\_c = np.array([c[0], c[1]])  
 radius\_c = c[2]  
 overlap = False  
  
 for fc in filtered\_circles:  
 center\_fc = np.array([fc[0], fc[1]])  
 radius\_fc = fc[2]  
  
 distance = np.linalg.norm(center\_c - center\_fc)  
 tolerance = +2 # 중복 기준 조정  
 # 중첩 여부 판단  
 if distance < (radius\_c + radius\_fc + tolerance):  
 overlap = True  
 break  
  
 if not overlap:  
 filtered\_circles.append(c)  
  
 bright\_count = 0  
 dark\_count = 0  
 brightness\_threshold = 128  
  
 for i in filtered\_circles:  
 center = (i[0], i[1])  
 small\_radius = i[2] // 4  
  
 # 중심의 작은 부분만 마스킹  
 mask = np.zeros(gray.shape, dtype="uint8")  
 cv2.circle(mask, center, small\_radius, 255, -1)  
 mean\_val = cv2.mean(gray, mask=mask)[0]  
  
 if mean\_val > brightness\_threshold:  
 bright\_count += 1  
 else:  
 dark\_count += 1  
  
 cv2.circle(image, center, i[2], (255, 0, 255), 2)  
  
 else:  
 bright\_count = 0  
 dark\_count = 0  
  
  
 return bright\_count, dark\_count  
  
  
def automatic\_perspective\_transform(image):  
 # 흑백 및 블러 처리  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 gray\_blurred = cv2.GaussianBlur(gray, (5, 5), 0)  
 thresh = cv2.adaptiveThreshold(  
 gray\_blurred, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY\_INV, 11, 2  
 )  
  
 # 엣지 검출과 닫힘 연산  
 edges = cv2.Canny(thresh, 50, 150)  
 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))  
 closed = cv2.morphologyEx(edges, cv2.MORPH\_CLOSE, kernel)  
  
 # 컨투어 찾기  
 contours, \_ = cv2.findContours(  
 closed, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE  
 )  
 if not contours:  
 print("윤곽을 찾을 수 없습니다.")  
 return None  
  
 # 가장 큰 사각형 찾기  
 contours = sorted(contours, key=cv2.contourArea, reverse=True)  
 for contour in contours:  
 epsilon = 0.02 \* cv2.arcLength(contour, True)  
 approx = cv2.approxPolyDP(contour, epsilon, True)  
 if len(approx) == 4:  
 pts = approx.reshape(4, 2)  
 rect = order\_points(pts)  
 # Check if rectangle is axis-aligned  
 if is\_rectangle\_axis\_aligned(rect):  
 # 이미지가 이미 평면이므로 변환하지 않음  
 return image  
 else:  
 # 투시 변환 적용  
 maxWidth, maxHeight = get\_max\_width\_height(rect)  
 dst = np.array(  
 [[0, 0], [maxWidth - 1, 0], [maxWidth - 1, maxHeight - 1], [0, maxHeight - 1]],  
 dtype="float32",  
 )  
 matrix = cv2.getPerspectiveTransform(rect, dst)  
 result = cv2.warpPerspective(image, matrix, (int(maxWidth), int(maxHeight)))  
 return result  
 else:  
 print("사각형을 찾을 수 없습니다.")  
 return None  
  
def is\_rectangle\_axis\_aligned(rect, angle\_threshold=5):  
 # 각 변의 각도를 계산  
 angles = []  
 for i in range(4):  
 pt1 = rect[i]  
 pt2 = rect[(i + 1) % 4]  
 dx = pt2[0] - pt1[0]  
 dy = pt2[1] - pt1[1]  
 angle = np.degrees(np.arctan2(dy, dx))  
 # Adjust angle to be between -90 and 90 degrees  
 angle = (angle + 90) % 180 - 90  
 angles.append(angle)  
 # 각도가 0 또는 90에 가까운지 확인  
 for angle in angles:  
 if not (abs(angle) < angle\_threshold or abs(abs(angle) - 90) < angle\_threshold):  
 return False  
 return True  
  
def order\_points(pts):  
 rect = np.zeros((4, 2), dtype="float32")  
 s = pts.sum(axis=1)  
 diff = np.diff(pts, axis=1)  
 rect[0] = pts[np.argmin(s)] # top-left  
 rect[2] = pts[np.argmax(s)] # bottom-right  
 rect[1] = pts[np.argmin(diff)] # top-right  
 rect[3] = pts[np.argmax(diff)] # bottom-left  
 return rect  
  
def get\_max\_width\_height(rect):  
 (tl, tr, br, bl) = rect  
 widthA = np.linalg.norm(br - bl)  
 widthB = np.linalg.norm(tr - tl)  
 maxWidth = max(int(widthA), int(widthB))  
 heightA = np.linalg.norm(tr - br)  
 heightB = np.linalg.norm(tl - bl)  
 maxHeight = max(int(heightA), int(heightB))  
 return maxWidth, maxHeight  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) != 2:  
 print("사용법: python hw1\_4.py <이미지 경로>")  
 else:  
 try:  
 bright\_count, dark\_count = get\_chess(sys.argv[1])  
 print(f"w:{bright\_count} b:{dark\_count}")  
 except TypeError:  
 print("get\_chess 함수에서 예기치 않은 반환값이 발생했습니다. 함수가 (bright\_count, dark\_count) 형식으로 반환하는지 확인하세요.")

설명 :

get\_chess는 주어진 이미지에서 원을 검출하고, 각 원의 밝기 값에 따라 밝은 원과 어두운 원의 개수를 계산하는 함수입니다.

구성 요소:

이미지 로딩 및 자동 투시 변환:

입력 이미지 파일을 불러와, automatic\_perspective\_transform 함수를 통해 평면화합니다.

그레이스케일 변환 및 블러 처리:

투시 변환된 이미지를 cv2.cvtColor로 그레이스케일로 변환하고, cv2.medianBlur로 노이즈를 줄여줍니다.

원 검출:

cv2.HoughCircles 함수로 원을 검출하며, 크기와 간격 조건에 맞는 원들을 탐지합니다.

원 중복 제거 및 필터링:

검출된 원을 반지름 크기에 따라 내림차순으로 정렬하고, 다른 원과 중복되는지 거리와 크기를 기준으로 판단해 중복된 원을 제거합니다.

밝기 측정 및 카운팅:

각 원의 중심에 작은 마스크를 적용해 평균 밝기 값을 측정하고, 밝기 값이 128 이상이면 밝은 원, 미만이면 어두운 원으로 분류하여 카운팅합니다.

automatic\_perspective\_transform는이미지에서 사각형 모양의 윤곽선을 자동으로 감지하고 원근을 보정하여 평면화하는 함수입니다..

구성 요소:

그레이스케일 및 블러 처리:

cv2.cvtColor로 그레이스케일로 변환한 뒤 cv2.GaussianBlur로 노이즈를 줄입니다.

이진화 및 엣지 검출:

cv2.adaptiveThreshold로 이진화하고 cv2.Canny를 사용해 엣지를 검출하며, morphologyEx의 닫힘 연산으로 윤곽을 강화합니다.

컨투어 탐지 및 사각형 감지:

cv2.findContours로 이미지 외곽선을 찾고, 가장 큰 사각형을 찾습니다. 사각형이 감지되면 좌표를 order\_points 함수로 정렬하고, 사각형이 축에 평행한지 is\_rectangle\_axis\_aligned로 확인합니다.

투시 변환:

이미지가 평면이 아니면 투시 변환을 적용해 사각형을 직사각형으로 변환합니다.

is\_rectangle\_axis\_aligned는 감지된 사각형이 축에 평행한지 확인하는 함수입니다..

작동 원리:

각 변의 각도를 계산하여, 각도가 0도 또는 90도에 가까운지 확인함으로써 평행 여부를 판단합니다.

order\_points는사각형의 네 좌표를 좌상단, 우상단, 우하단, 좌하단 순으로 정렬하는 함수입니다..

작동 원리:

좌표의 합(s)과 차이(diff)를 기준으로 위치를 구분하여 정렬합니다.

get\_max\_width\_height는 변환된 이미지의 최대 폭과 높이를 계산하는 함수입니다.

작동 원리:

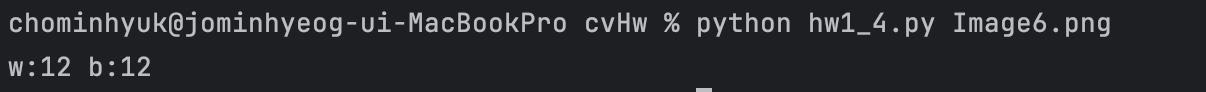
각 변의 길이를 계산하여, 가장 긴 폭과 높이를 선택합니다.

결과 :

해당 이미지는 Image8.png로 아래와 같습니다.

체커, 보드게임, 실내 게임 및 스포츠, 테이블 게임이(가) 표시된 사진

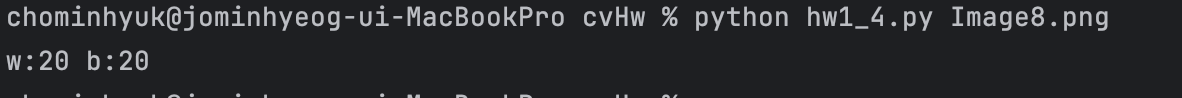
자동 생성된 설명



해당 이미지는 Image8.png로 아래와 같습니다.

실내 게임 및 스포츠, 테이블 게임, 보드게임, 게임이(가) 표시된 사진

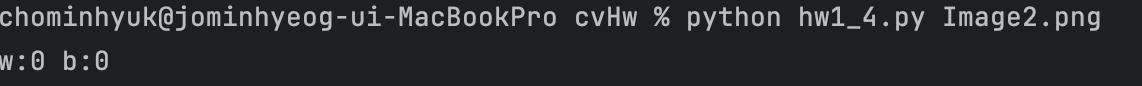
자동 생성된 설명



해당 이미지는 Image2.png로 결과는 아래와 같습니다.

사각형, 체스, 직사각형, 보드게임이(가) 표시된 사진

자동 생성된 설명



해당 이미지는 Image5.png로 결과는 아래와 같습니다.

체커, 실내 게임 및 스포츠, 보드게임, 테이블 게임이(가) 표시된 사진

자동 생성된 설명

