**컴퓨터 비전**

Assign#1. Ar cube 구현하기

|  |
| --- |
| 스 쿨 : ETRI 스쿨 |
| 이 름 : 김 기 범 |
| 학 번 : 02321078 |

1. 서론

AR (Augmented Reality) 기술은 현실 세계에 가상의 객체를 합성하여 현실과 가상의 경계를 무너뜨립니다. 이번 과제는 OpenCV를 사용하여 라이브 카메라 영상에서 마커(체커보드)를 인식하고, 이를 기반으로 인식된 마커 위에 가상의 AR 큐브를 그려주는 프로그램을 구현하는 것입니다.

1. 본론
2. 주요 코드
3. 체커보드 꼭지점 인식 및 3D 좌표 생성

# 체커보드의 꼭지점 수 설정

corner\_x = 4

corner\_y = 4

# 3D 좌표를 저장할 배열

objpoints = []

imgpoints = []

# 체커보드의 꼭지점 좌표 생성

objp = np.zeros((corner\_x\*corner\_y, 3), np.float32)

objp[:,:2] = np.mgrid[0:corner\_x, 0:corner\_y].T.reshape(-1, 2)

corner\_x와 corner\_y는 체커보드의 가로 및 세로 꼭지점 수를 설정합니다. 여기서는 4\*4의 체커보드로 설정하였습니다.

objpoints와 imgpoints는 체커보드의 3D 객체 및 2D 이미지 꼭지점 좌표를 저장하는 배열입니다.

objp는 체커보드의 3D 좌표를 생성하며, 가로와 세로 인덱스를 사용하여 모든 꼭지점의 3D 좌표를 초기화합니다.

1. 카메라 캘리브레이션

while True:

ret, frame = cap.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

ret, corners = cv2.findChessboardCorners(gray, (corner\_x, corner\_y), None)

if ret == True:

if len(imgpoints) <= 20:

objpoints.append(objp)

imgpoints.append(corners)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

무한 루프를 사용하여 카메라에서 프레임을 읽고, 각 프레임을 흑백으로 변환하여 gray에 저장합니다.

cv2.findChessboardCorners 함수를 사용하여 체커보드의 코너를 찾고 corners에 저장합니다.

코너를 찾으면, objpoints에는 3D 좌표, imgpoints에는 2D 이미지 좌표를 저장합니다.

만약 objpoints와 imgpoints에 충분한 데이터가 있다면, cv2.calibrateCamera 함수를 사용하여 카메라 행렬(mtx)과 왜곡 계수(dist)를 추정합니다.

이때 카메라 행렬과 왜곡계수를 미리 알고 있으면 좋겠으나, 그렇지 않은 경우가 대부분이기에 실시간으로 계속해서 cv2.calibrateCamera 함수를 실행합니다.

다만, 그 경우 연산량이 많아지게 되기 때문에 20번정도만 cv2.calibrateCamera 함수를 실행하고, 이후에는 최종 카메라 행렬, 왜곡 계수, 회전 벡터, 변환 벡터를 이용하도록 코드를 작성하였습니다.

1. AR 큐브를 이미지 평면에 투영하는 함수

# 점들을 투영하는 함수

def project\_points(axis, rvec, tvec, mtx, dist):

# 회전 벡터를 회전 행렬로 변환한다

R, \_ = cv2.Rodrigues(rvec)

# 외부 행렬 [R | t] 구성한다

RT = np.hstack((R, tvec))

# 투영 행렬 P = K [R | t]를 계산한다

P = np.dot(mtx, RT)

# 3D 점들을 2D로 투영한다

imgpts = []

for point in axis:

# 점을 homogeneous 좌표로 확장한다

point\_homogeneous = np.append(point, 1)

# 투영 행렬을 사용하여 점을 투영한다

imgpt\_homogeneous = np.dot(P, point\_homogeneous)

# 픽셀 좌표로 다시 변환한다

imgpt = imgpt\_homogeneous[:2] / imgpt\_homogeneous[2]

imgpts.append(imgpt)

return np.array(imgpts)

project\_points 함수는 주어진 3D 점들(axis)을 2D 이미지 평면에 투영하는 작업을 수행합니다.

이 함수는 주어진 카메라의 회전 벡터(rvec), 변환 벡터(tvec), 카메라 행렬(mtx), 및 왜곡 계수(dist)를 사용하여 이 투영을 수행합니다.

Rodrigues 함수를 이용해서 회전 벡터(rvec)를 회전 행렬(R)로 변환합니다.

이후, 회전 행렬과 변환 벡터(tvec), 카메라 행렬(mtx)을 이용해서 투영 행렬(P)을 계산합니다.

투영 행렬(P)을 이용해서 3D 큐브의 꼭지점 좌표를 2D 이미지 평면에 투영해서 픽셀 좌표(imgpt)를 구합니다.

1. AR 큐브 그리는 함수

# AR 큐브를 그리는 함수

def draw\_cube(img, corners, imgpts):

imgpts = np.int32(imgpts).reshape(-1, 2)

# Draw the ground in green

img = cv2.drawContours(img, [imgpts[:4]], -1, (0, 255, 0), -3)

# Draw pillars in blue color

for i, j in zip(range(4), range(4, 8)):

img = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]), (255, 0, 0), 3)

# Draw top in red color

img = cv2.drawContours(img, [imgpts[4:]], -1, (0, 0, 255), 3)

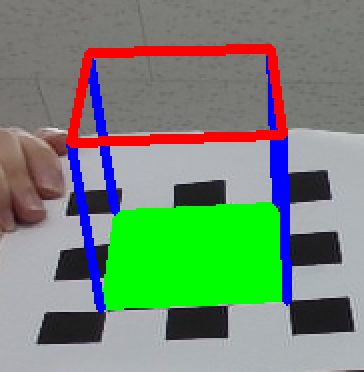
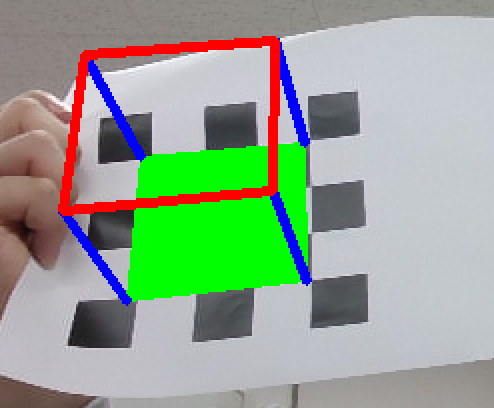
return img

draw\_cube 함수는 체커보드 꼭지점과 큐브의 이미지 꼭지점을 사용하여 AR 큐브를 그리는 함수입니다.

큐브의 바닥 면은 초록색, 기둥은 파란색, 상단은 빨간색으로 그려집니다.

img는 입력 이미지, corners는 체커보드의 꼭지점 좌표, imgpts는 큐브의 2D 이미지 꼭지점 좌표입니다.

1. 결과 캡쳐



코드를 테스트 한 결과, 체커보드의 움직임에 따라 잘 추적하는 모습을 보입니다.

1. 결론

이 프로젝트를 통해 AR Cube를 구현하고 라이브 카메라 영상에서 마커(체커보드)를 인식하고 가상의 AR 큐브를 그리는 것을 학습할 수 있었습니다. 수업에서 배운 이론적인 내용을 실제 코드로 구현하는 과정을 통해 실제 프로젝트에 적용하고 결과를 얻을 수 있었기 때문에 매우 의미 있었습니다.

또한, OpenCV 라이브러리는 사용해 본 적이 있었으나, 일부 이미지를 처리하는 정도의 간단한 수준의 코드들 만을 사용해 보았으나, 이번 기회를 통해 이미지 처리와 카메라 인식을 수행하는 과정을 터득하였으며, Python을 이용하여 간편하게 프로토타입을 개발할 수 있는 능력을 향상시킬 수 있었습니다.

1. Github 주소

<https://github.com/chroion/computer-vision/blob/main/HW%231/assign%231.py>