Camera Calibration 을 통한 3 차원 좌표 추출

- openCV 를 활용한 영상처리 프로젝트 -

이름: 성수호

Problem 1: 촬영된 영상을 보정(calibration) 작업 후 파라미터 값을 통해 3 차원 좌표 추출

1. 문제의 개요

본 프로그램을 간략히 설명하면 다음과 같다.

- 체스보드 코너점을 통해 카메라 파라미터의 값을 갖는다.
- 카메라로 프로젝트에 사용할 영상의 첫 이미지를 입력한다.
- 영상 속 물체의 크기 정보를 통해 카메라로부터 실제 거리를 계산 및 출력한다.

2. 알고리즘

본 프로그램 작성을 위한 알고리즘을 Pseudo 코드 형태로 나타내면 다음과 같다.

2.1. Calibration

- 체스보드 이미지를 통해 코너점 검출.
- 객체 지점과 이미지 지점으로 카메라 보정작업을 통해 파라미터 추출

2.2. Object 검출

- 입력된 영상에서 물체와 배경 사이의 테두리(edge) 검출
- 객체의 테두리 정보로 윤곽선 검출 및 최소 면적 사각형 그리기
- box 를 그리고 사각형의 각 꼭지점 좌표를 통해 중심점 출력

2.3. 실제거리 구하기

- 카메라 보정 매트릭스 정보를 통해 이미지 센서의 물체크기를 출력
- 초점거리 비례하여 실체 물체의 크기와 이미지 센서의 물체크기 정보를 통해 카메라와 물체 사이의 실제 거리를 출력

3. 프로그램 구조 및 설명

3.1. 체스보드 패턴 검출 및 Calibration

In [1]:
```
import numpy as np
import cv2
import glob
import matplotlib.pyplot as plt
%matplotlib qt
```

In [2]:
```
objp = np.zeros((6*8,3), np.float32)
# print(objp)
objp[:,:2] = np.mgrid[0:8, 0:6].T.reshape(-1,2)

objpoints = [] # 실세계 3차원 지점(객체지점)
imgpoints = [] # 이미지평면 2차원 이미지 지점
```

In [3]:
```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001

images = glob.glob('calibration_wide/GO*.jpg')
for idx, fname in enumerate(images):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ret, corners = cv2.findChessboardCorners(gray, (8,6), None)

    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners,(5,5),(-1,-1),criteria)
        imgpoints.append(corners2)#이미지 지점 입력
        cv2.drawChessboardCorners(img, (8,6), corners2, ret)
cv2.destroyAllWindows()
```

In [4]:
```
t, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
                                                 imgpoints,
                                                 gray.shape[::-1],
                                                None,None)
print(mtx)
```

- 체스보드 패턴을 찾기 위해, cv2.findChessboardCorners() 함수를 사용하여 코너지점과 패턴이 발견되었는지의 여부를 반환. 본 프로젝트에서는 8*6 격자를 사용한다.
- 코너를 발견하면, cv2.cornerSubPix() 함수를 사용하여 정확도를 높인다. 또한 cv2.drawChessboardCorners() 함수를 사용해 코너 결과를 그릴 수

있다.

- 패턴 검출을 통해 객체 지점(objpoints)과 이미지 지점(imgpoints)을 파악했으므로, cv2.calibrateCamera() 함수를 이용해 카메라 매트릭스를 구한다. 이 함수는 카메라 메트릭스, 왜곡 계수, 회전/이동 벡터 등이 반환된다.

3.2. 추적 Object 검출 및 Bounding Box 코너 좌표 반환.

```
from imutils import paths
import numpy as np
import imutils
```

```
gray = cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray,(5,5),0)
edged = cv2.Canny(gray, 70, 180)

cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIM
cnts = imutils.grab_contours(cnts)
c = max(cnts, key = cv2.contourArea)
marker = cv2.minAreaRect(c)

#bbox그리기
box = cv2.boxPoints(marker) if imutils.is_cv2() else cv2.boxPoints(marker
box = np.int0(box)
#좌측상단부터 반시계방향 좌표
cor1, cor2, cor3, cor4 = box
centerX,centerY = ((cor1 + cor4)/2 + (cor2 + cor3)/2)/2
cx,cy = int(centerX),int(centerY)
print(centerX,centerY)
cv2.drawContours(src, [box], -1, (0,255,0), 2)
cv2.circle(src,(cx,cy), 1, (0,255,0), 2)


cv2.imshow('src',src)
cv2.waitKey()
cv2.destroyAllWindows()
```

- 이미지를 그레이스케일로 전환하고, 블러 처리를 통해 고주파 잡음을 제거한다. 그리고 cv2.Canny()를 통해 테두리를 검출 한다.

- Cv2.findContours 함수로 테두리 중 마커를 찾은 다음 가장 큰 영역을 가진 윤곽선을 결정한다.
- 윤곽선을 따라 최소 면적의 사각형(bbox)을 그린다.
- 사각형의 각 꼭지점으로부터 좌표 값을 변수로 반환하고, 중심점을 계산한다. 나중에 좌표 값을 통해 너비 픽셀 값을 구한다.

3.3. 실제거리 구하기

```
#초점거리 구하기
exif_img = PIL.Image.open(images[0])
exif_data = {
    PIL.ExifTags.TAGS[k]:v
    for k, v in exif_img._getexif().items()
    if k in PIL.ExifTags.TAGS}
```

```
fx = mtx[0][0]
fy = mtx[1][1]


m = ((fx+fy)/2) / 3.0 # 단위 : px/mm
print('m:',m,'fx:',fx,'fy:',fy)


x = 1080*m/1280
print('x :',x)


objectWidth = np.sqrt((cor4[0]-cor1[0])**2 + (cor4[1]-cor1[1])**2)
imgSensor = objectWidth/x
print('objectWidth : ',objectWidth)
print('imgSensor : ',imgSensor)
```

```
186.9178605147101 fx: 560.2255661908922 fy: 561.2815968973683
: 157.71194480928665
jectWidth :  633.7586291325745
gSensor :  4.0184567497341295
```

```
# 카메라에서 개체까지의 거리
dist = 380 * 3.0 / imgSensor
```

- ExifTags 라이브러리로 프로젝트에서 사용된 카메라 정보를 가져온다.
- Fx 와 Fy 는 카메라 매트릭스에서 가져온 값이며 초점 길이에 배율 인수를 곱한 값이다. 따라서 카메라 정보에서 초점길이를 얻어와 배율 인수를 구할 수 있다.
- 저해상도의 배율인수(x)는 고해상도의 배율인수를 비례하여 계산한다
- 3.2 에서 최소사각형을 통해 구한 너비 픽셀 값을 저해상도 배율인수로 나눠 이미지센서의 물체크기를 구한다.
- 실제 크기 : 이미지센서의 물체크기 = 카메라와 물체 사이 실제거리 : 초점 거리의 비례 공식을 통해서 실제 거리를 구할 수 있다.

4. 결론

- 카메라 보정 작업에서 얻어지는 카메라 매트릭스와 내부 정보를 통해 3 차원 좌표를 추출하는 작업을 수행했다. 본 프로젝트에 앞서 만들었던 ball tracking 코드와 병합한다면 실시간으로 공의 이동 거리나 속도, 속력 등을 3 차원 그래프로 표현할 수 있을 것 같다. 또한 스포츠 중계화면에서 보여주는 미니맵처럼 X, Z 좌표를 이용해 스카우팅뷰(scouting view)도 표현할 수 있다.

5. 개선방향

- Calibration 작업에서 외부파라미터에 대한 이해가 부족했다. 한 영상처리 전문가에 따르면 외부파라미터 셋업을 통해 직접적인 객체의 실제 정보를 구하지 않아도 영상 속 객체들의 3 차원 정보를 가져올 수 있다고 한다. 앞으로 이부분에 대한 깊이 있는 연구가 필요하다.

- 빨간 공을 트래킹 하는 건 HSV 를 이용했기 때문에 같은 색상의 물체가 겹치거나 다른 색상을 가진 물체가 가리는 경우가 생기면 물체에 대한 좌표에 에러가 발생한다. 그래서 딥러닝을 통해 학습된 데이터로 객체를 인식하는 방법으로 이러한 에러 발생을 줄이고 정확한 좌표 추출하는 작업이 필요하다.