ArUCo Marker

ai-contents

Exported on 06/18/2023

Table of Contents

1 ArUCo Marker의 개념

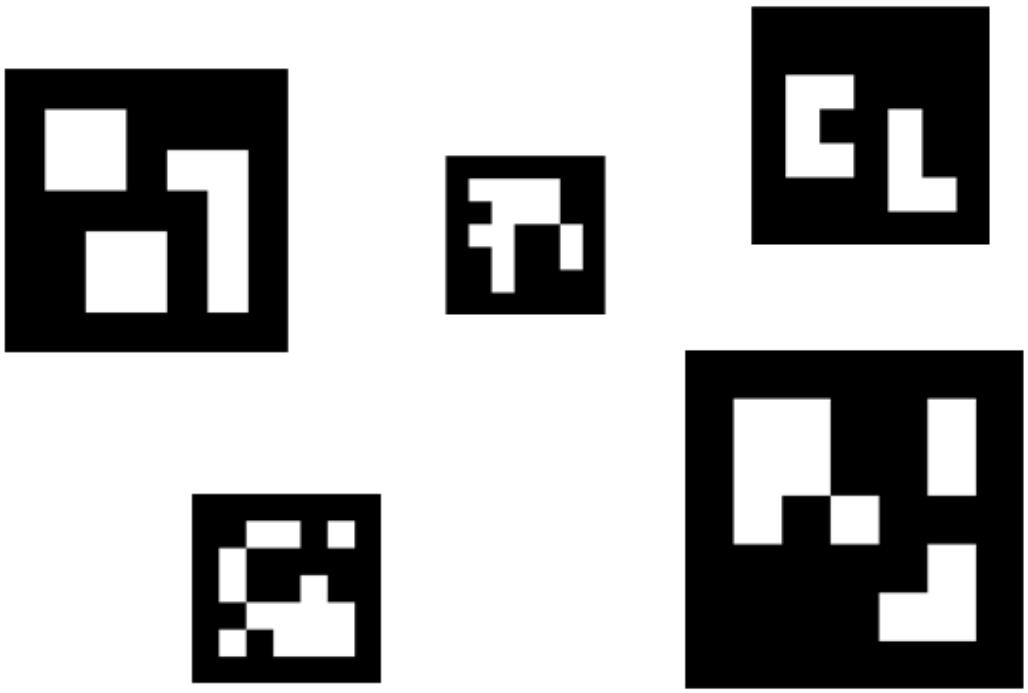
1.1 1. 기준 마커

- 일정한 포맷으로 만들어진 인공적인 랜드마크
- N x N 크기
- 정사각형 형태

1 테두리를 1로 설정(왼), 테두리를 2로 설정(오)

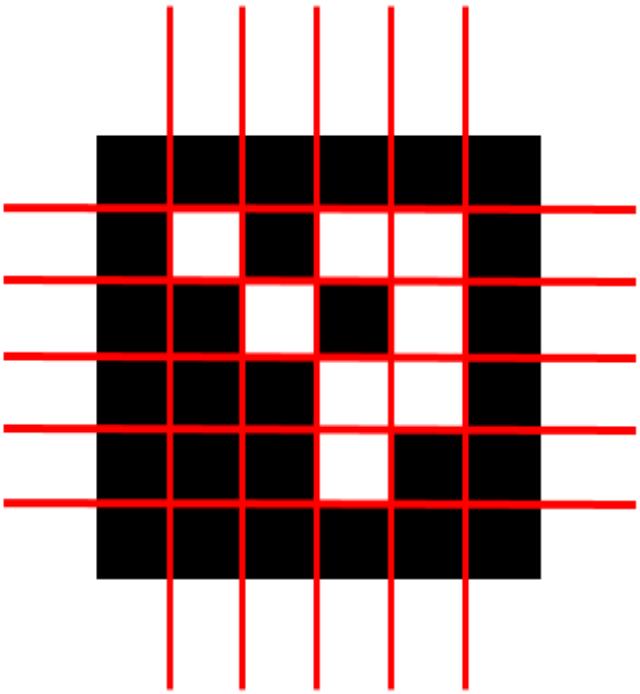
1.2 2. 2차원의 비트 패턴

- 흰색 셀과 검은색 셀의 조합
- 마커의 고유 ID를 표현
- 마커를 식별하는데 사용



2-1. ArUCo 마커에서 N x N이 의미하는 것

- 2차원 비트 패턴의 크기
- DICT_4X4의 경우, 가로 4, 세로 4인 격자 내에서 존재할 수 있는 16개의 셀을 사용하여 마커 ID를 표현
- 실제로는 검은색 테두리 영역까지 포함해서 6X6으로 생성



2 흰 부분은 1, 검은 부분은 0의 형태로 입력

1.3 3. ArUCo 마커를 활용한 3차원 자세 추정

- 이미지 내에 있는 마커를 검출
- 4개의 코너에 대한 좌표를 구함
- 해당 코너 좌표와 카메라 캘리브레이션을 활용하여, 카메라 좌표계 기준의 마커의 3차원 자세 추정

2 ArUCo 마커를 활용한 3차원 자세 추정 알고리즘

2.1 1. 3차원 자세 추정 알고리즘 개요

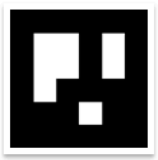
2.1.1 1-1. ArUCo 마커 이미지 업로드

- 3차원 자세 추정에 사용할 ArUCo 마커의 이미지를 업로드
- 이미지 뿐만 아니라 영상, 웹캠도 가능

2.1.2 1-2. ArUCo 마커의 유형에 대한 딕셔너리 업로드

- ArUCo 마커의 유형을 딕셔너리 형태로 정의
- 딕셔너리란?
    - Key와 Value 쌍으로 구성된 데이터
    - 이름-홍길동, 생일-몇월 몇일 등의 대응 관계를 표현
- ArUCo마커에 대한 딕셔너리 예시 일부

```
ARUCO_DICT = {
        "DICT_4X4_100": cv2.aruco.DICT_4X4_100,
        "DICT_5X5_100": cv2.aruco.DICT_5X5_100,
        "DICT_6X6_100": cv2.aruco.DICT_6X6_100,
        "DICT_7X7_100": cv2.aruco.DICT_7X7_100
        }
```



DICT_4X4_100_id_23.png

DICT_5X5_100_id_23.png

DICT_6X6_100_id_23.png

DICT_7X7_100_id_23.png

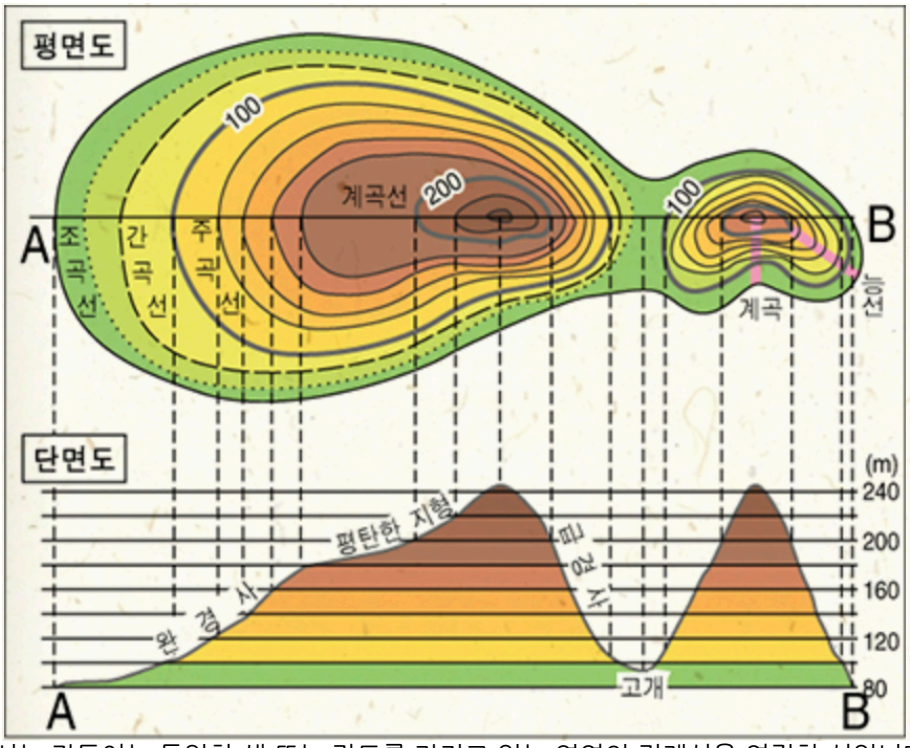
2.1.3 1-3. ArUCo 마커 검출에 대한 파라미터 정의

- ArUCo 마커 검출을 위해 필요한 파라미터는 다양하게 존재
    - adaptiveThreshWinSizeMin
    - adaptiveThreshWinSizeMax
    - adaptiveThreshWinSizeStep
    - 이 외에도 총 29종류의 파라미터(위 파라미터 포함) 존재

- 하지만 우리는, openCV에서 정의해준 디폴트 값을 사용해도 됩니다.

2.1.4 1-4. ArUCo 마커 검출 수행

- ArUCo 마커를 검출해서, 3차원 자세 추정에 필요한 정보인 코너 좌표를 추출합니다.
  - 코너 좌표 : ArUCo의 4개의 꼭짓점에 대한 좌표
- 마커 검출에 필요한 핵심 기술은 칸투어(Contour)입니다.
  - 칸투어는 '등고선'이라는 뜻으로 같은 값을 가진 곳을 연결한 선입니다.



- 이미지에서는 칸투어는 동일한 색 또는 강도를 가지고 있는 영역의 경계선을 연결한 선입니다.
- 물체의 모형 분석이나 객체 감지 및 인식 알고리즘에 주로 사용됩니다.
- 칸투어 알고리즘에 근사화 알고리즘을 적용하면, 아래와 같이 마커의 코너 좌표를 얻을수 있습니다.
- 이 또한 openCV의 cv2.aruco.detectMarkers 함수를 사용해서, 간단하게 구현할 수 있습니다.

2.1.5 1-5. 3차원 자세 추정

- 이제 ArUCo 마커의 코너 좌표를 활용해서 3차원 자세를 추정할 것입니다.
- 이 역시 openCV의 cv2.aruco.estimatePoseSingleMarkers 함수를 사용해서 간단하게 구현할 수 있습니다.
- 위 함수를 활용한 3차원 자세 추정을 위해, 필요한 카메라의 파라미터는 다음과 같습니다.
  - 내부 카메라 행렬
  - 렌즈 왜곡 계수
- 해당 정보는 카메라의 파라미터를 추정하는 과정인 카메라 캘리브레이션을 통해 구할수 있습니다.

    - 3차원 점들이 영상에 투영된 위치를 구하거나 역으로 영상좌표로부터 3차원 공간좌표를 복원할 때에는 다양한 내부 요인(ex. 렌즈와 센서의 기구적인 부분, 렌즈와 센서의 거리 기타 등)을 제거해야만 정확한 계산이 가능해집니다.
    - 이러한 내부 요인의 파라미터 값을 구하는 과정을 카메라 캘리브레이션이라 부릅니다.
- 이 정보들을 활용하면 ArUCo마커의 3차원 자세를 추정할 수 있습니다.
- 여기까지 ArUCo 마커를 활용한 3차원 자세 추정 알고리즘에 대한 개요입니다.

2.2 2. OpenCV를 활용한 카메라 캘리브레이션

- 카메라 좌표계 기준의 ArUCo 마커의 3차원 자세추정을 위해 카메라 캘리브레이션이 필요하고 이를 통해 내부 카메라 행렬과 렌즈 왜곡 계수를 구해야 한다고 했습니다.
- 이에 대한 이해를 위해, 카메라 캘리브레이션에 대해 조금 더 알아보겠습니다.

2.2.1 2-1. 카메라 캘리브레이션의 개요

- 우리는 '월드(실제 세상) 좌표계'를 '카메라 좌표계'로 변환하기 위해, 변환 행렬을 구해야 합니다.
- 이때 변환 행렬은 '내부 카메라 행렬'과 '외부 카메라 행렬'로 구성되어 있습니다.
- 각각 알아 보자면
    - 내부 카메라 행렬
        - 카메라 파라미터에 대한 3X3 행렬
        - 초점 거리, 이미지 중심 좌표, 축 사이의 기울기 등의 파라미터로 구성된 행렬입니다
    - 외부 카메라 행렬
        - 3X1 회전 행렬과 3X1 이동 벡터의 조합
- 위 변환 행렬을 구하는 것이 카메라 캘리브레이션의 목적입니다.

2.2.2 2-2. 카메라 캘리브레이션 알고리즘의 입출력

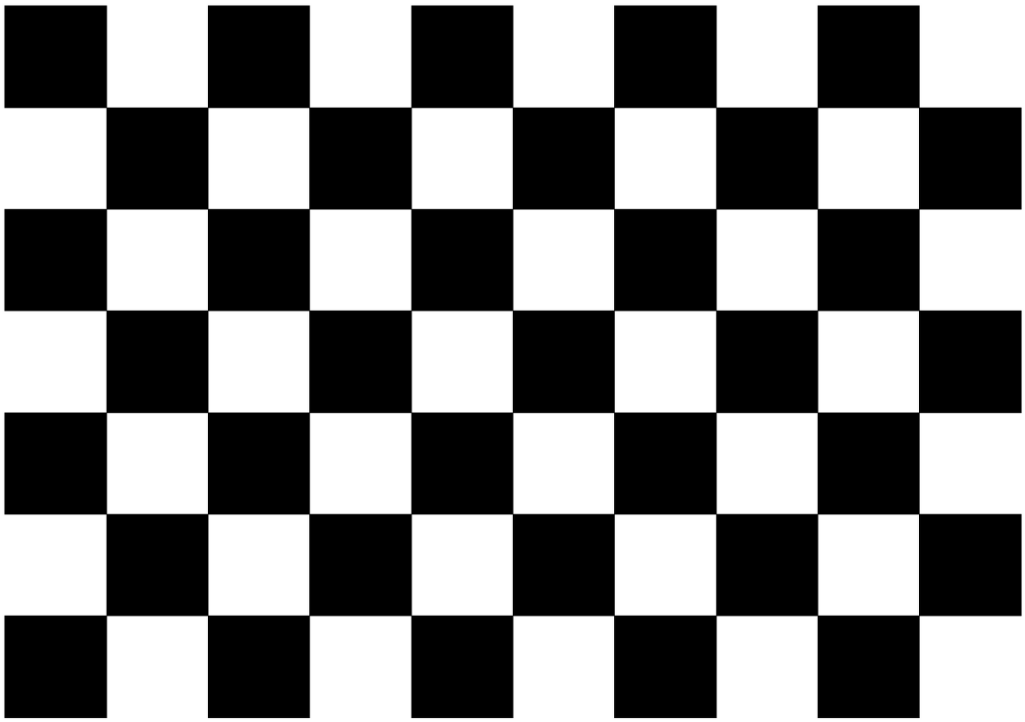
- 입력
    - 다양한 각도 및 위치에서 촬영한 체커보드 이미지
        - 이를 통해 체커보드 코너에 대한 2D 이미지 좌표 및 3D 월드 좌표를 구할수 있다.
        - 이때, 본인이 사용할 카메라를 가지고 카메라 캘리브레이션을 진행해야 한다.
            - ex. 핸드폰 카메라로 체커보드를 찍은 후 웹캠으로 ArUCo 마커를 이용한 3차원 자세 추정을 하면 무의미
- 출력
    - 내부 카메라 행렬, 렌즈 왜곡 계수, 회전 벡터(3X1), 이동 벡터(3X1)

2.2.3 2-3. 카메라 캘리브레이션 방법

1. 체커보드 패턴으로 월드 좌표 정의
    a. 체커보드 패턴은 이미지에서 뚜렷하고 쉽게 감지할 수 있기에 위치 파악에 이상적입니다.
    b. 이에 따라 캘리브레이션을 위한 이미지로 자주 사용됩니다.
    c. 체커보드 패턴에서 코너의 좌표를 찾는 것이 핵심입니다.
    d. 책상 혹은 벽 등에 체커보드 이미지의 위치를 고정시킵니다.
    e. 이렇게 월드(실제 세상)에 고정된 체커보드가 월드 좌표계가 됩니다.
    f. 체커보드의 모든 코너는 월드 좌표계의 원점으로 선택할 수 있습니다.
    g. 체커보드의 Z좌표는 0이므로(체커보드가 평면이기 때문), 체커보드의 모든 점은 XY평면에 있다는 특징이 있습니다.
2. 다양한 시점에서 체커보드를 여러장 촬영
    a. 보다 정확한 카메라 캘리브레이션을 위해 다양한 시점에서 체커보드 패턴을 촬영합니다.
    b. 일반적으로 7~10장 가량의 체커보드 이미지를 촬영하면 됩니다.
3. 체커보드의 2D좌표 찾기
    a. 이미지에서 체커보드 코너의 픽셀 위치 또한 openCV 함수인 findChessboardCorners를 사용하면, 2D 좌표를 손쉽게 구할 수 있습니다.

```
ret, corners = cv2.findChessboardCorners(image, patternSize, None)
'''
입력
  image는 체커보드 사진,
  patternSize는 체커보드 격자 형태

출력
  corners : 감지된 격자 형태
'''
```



1. 이제 코너의 좌표를 더욱 정교하게 얻어야 합니다. 이는 openCV의 cornerSubPix라는 함수를 통해 구할수 있습니다.

```
corners2 = cv2.cornerSubPix(image, corners, winSize, zeroZone, criteria)
'''
입력
  image는 체커보드 사진,
  corners는  findChessboardCorners함수를 통해 구한 체커보드 코너의 픽셀 위치

출력
  corners2 : 정교해진 코너의 픽셀 위치
'''
```

2. 카메라 캘리브레이션
   a. 이제 카메라 캘리브레이션을 할 수 있습니다. openCV의 calibrateCamera함수를 통해 카메라 파라미터 정보를 받습니다.
   b. 해당 함수는 'A Flexible New Technique for Camera Calibration(1998)' 논문을 기반으로 구현됩니다.

```
ret, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(
                                        objectPoints, imagePoints,
imageSize)
```

```
'''
입력
  objectPoints는  3D점 벡터로 구성된 벡터,
  imagePoints는  2D이미지 점 벡터로 구성된 벡터
  imageSize는 이미지의 크기

출력
  cameraMatrix는 내부 카메라 행렬,
  distCoeffs는 렌즈 왜곡 계수,
  rvecs는 회전 벡터(3X1),
  tvecs는 이동 벡터(3X1)
'''
```

- 여기까지가 카메라 캘리브레이션에 대한 전반적인 내용입니다.
- 다음 강의에서 openCV를 활용한 실습을 통해 ArUCo 마커의 3차원 자세 추정을 실습해보겠습니다.