

Measurement Volumetric

TEAM 4조 (최강 4조) 🏆

최호준, 성우진, 이형구, 박찬수, 오의택

리테일앤인사이트

Contents

01 프로젝트 소개

- 팀원 소개 및 역할
- 프로젝트 주제

02 프로젝트 구현 과정

- 카메라 촬영조건
- 카메라 보정(Calibration)
- 배경제거
- 객체 탐지(Object Detection)
- 외곽선 검출(findContours)
- 체적 도출

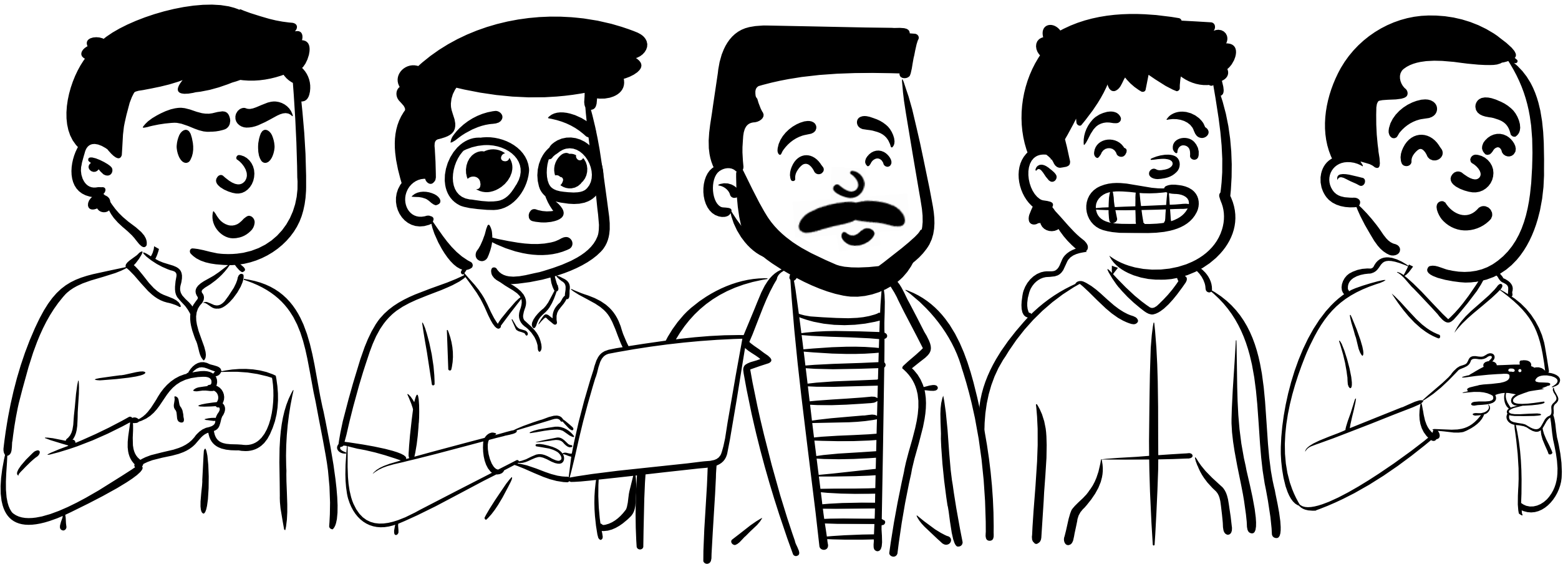
03 실측 테스트 결과

- 정량 평가 기준 & 결과

04 데모 시연 & 후기

※ 부록 첨부

1. 프로젝트 소개



최호준

가로, 세로 구하기
원기둥 측정
코드 정리

성우진

높이 구하기
실세계 좌표계 규명

이형구

객체 탐지 및 실험
그래프를 통한 시각화

박찬수

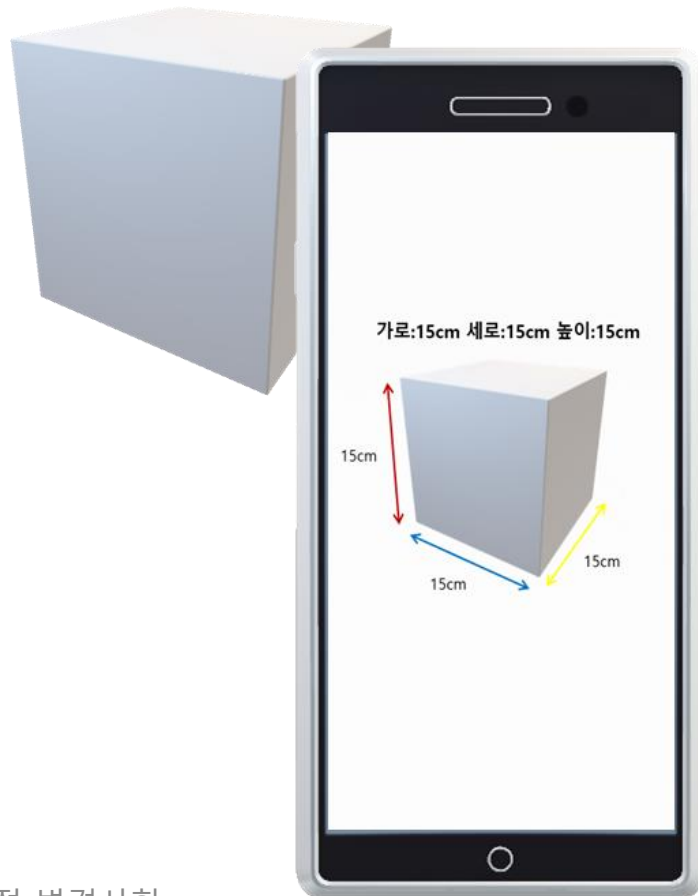
사진 촬영 및 실험
외각선 검출

오의택

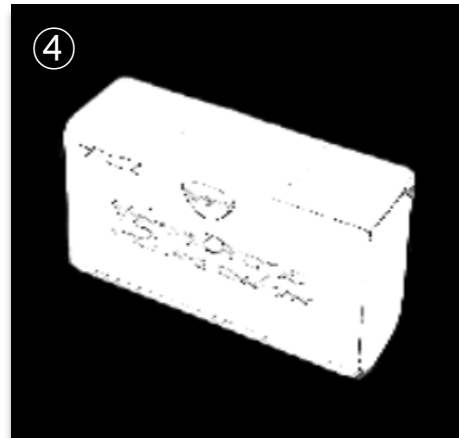
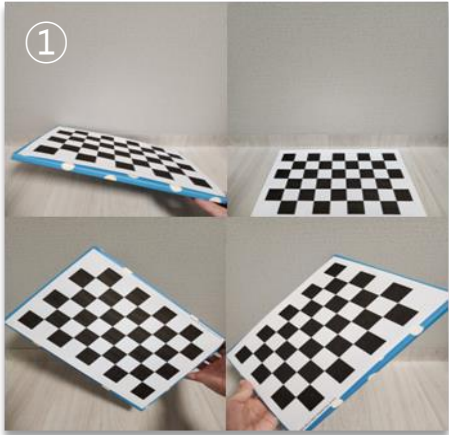
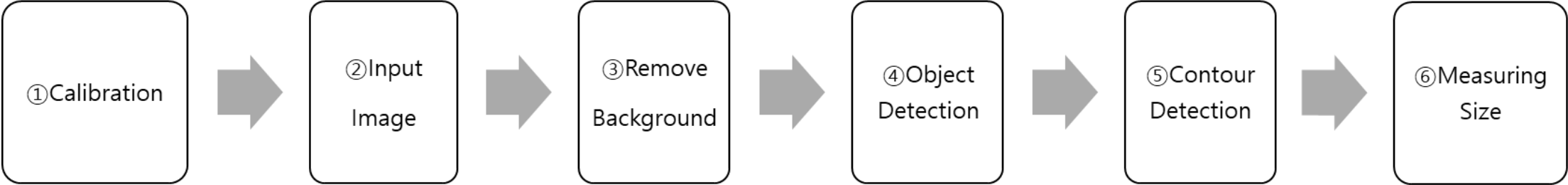
객체 탐지 및 실험
발표 자료 구성

•프로젝트 주제

한 장의 이미지로부터 상품 체적정보(가로, 세로, 높이)를 구하는 프로그램 개발 - 오차범위 5mm이내



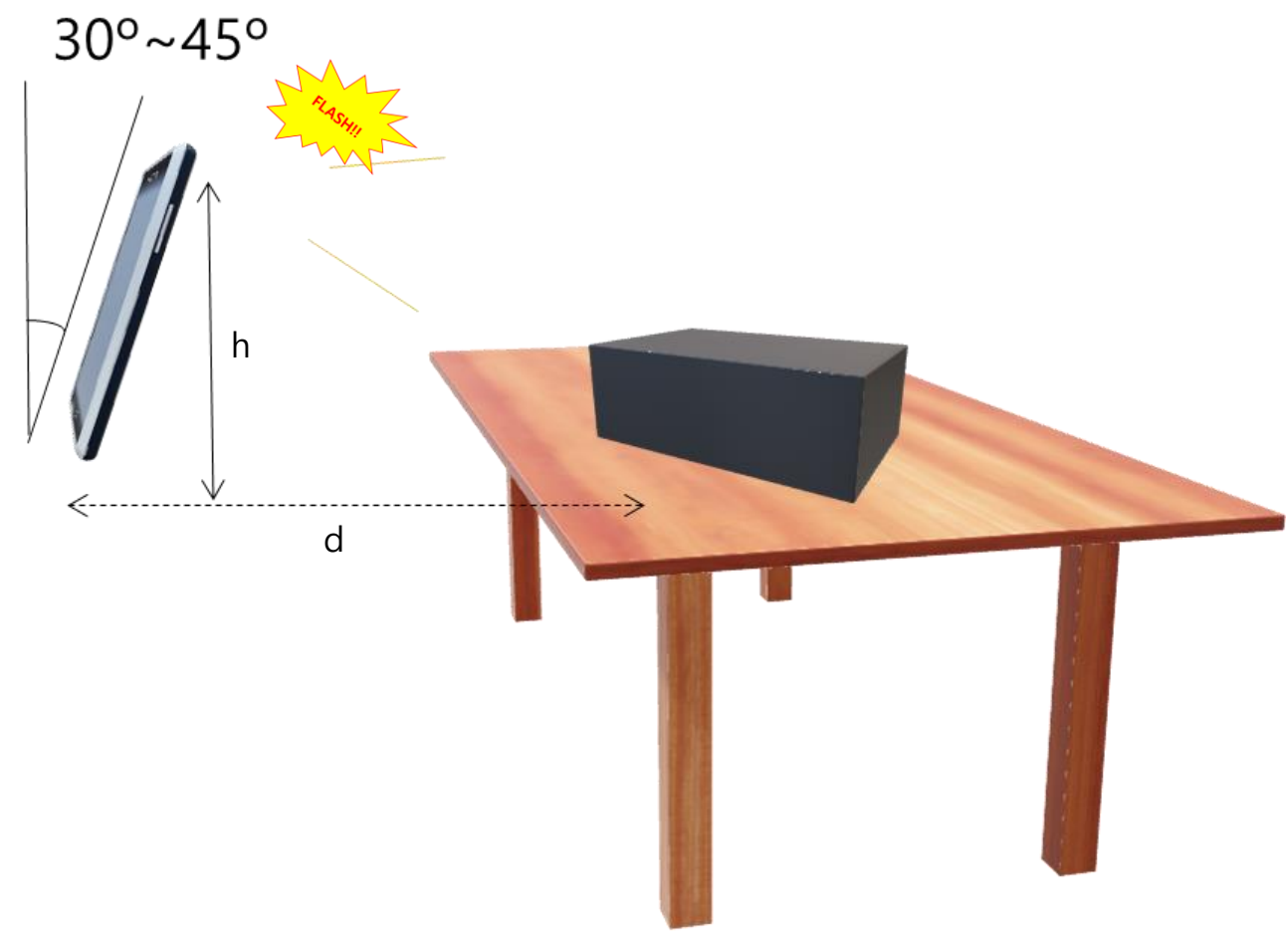
Overview



2.1 카메라 촬영 환경

카메라 촬영 환경

촬영 환경



물체의 3면이 보인다



물체의 3면이 보이지 않는다



카메라의 높이가 높다



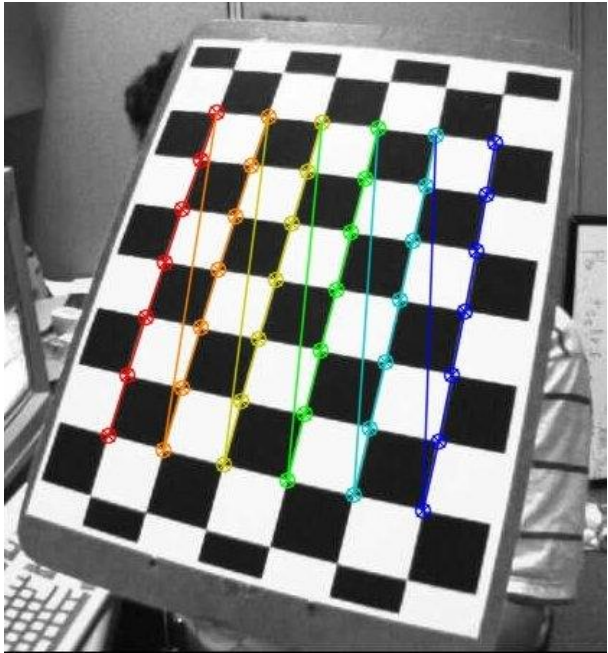
2.2 카메라 보정 (*Calibration*)

2.2

카메라 보정(Calibration)

- 카메라 보정을 통해 체커보드 인식 및 물체 측정에 필요한 정보 획득
 - Intrinsic 행렬, 왜곡 계수, Rotation 벡터, Translation 벡터, World 좌표계

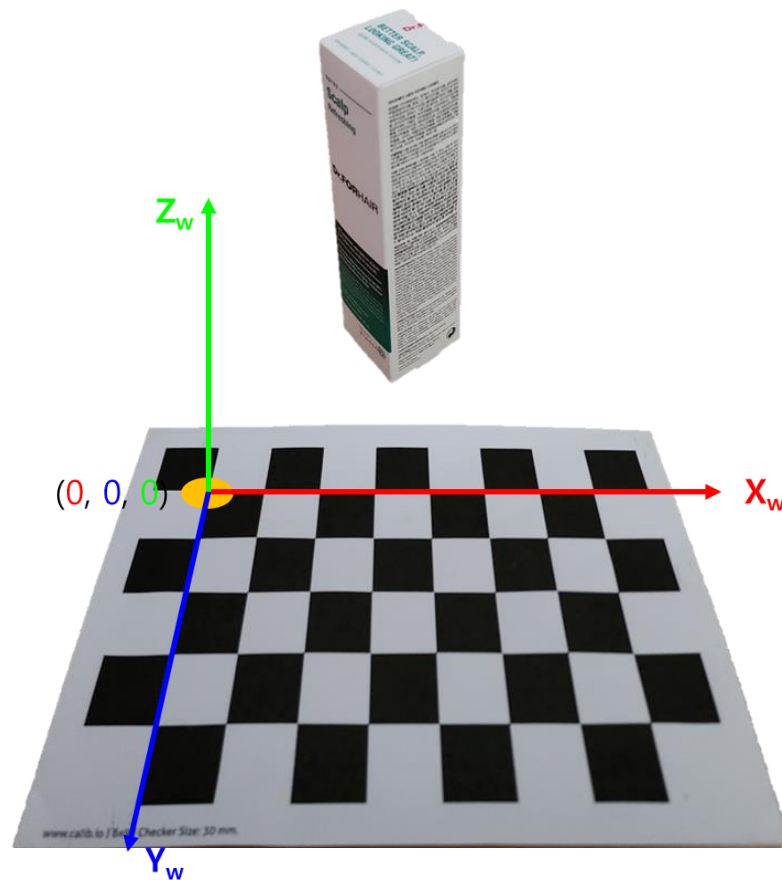
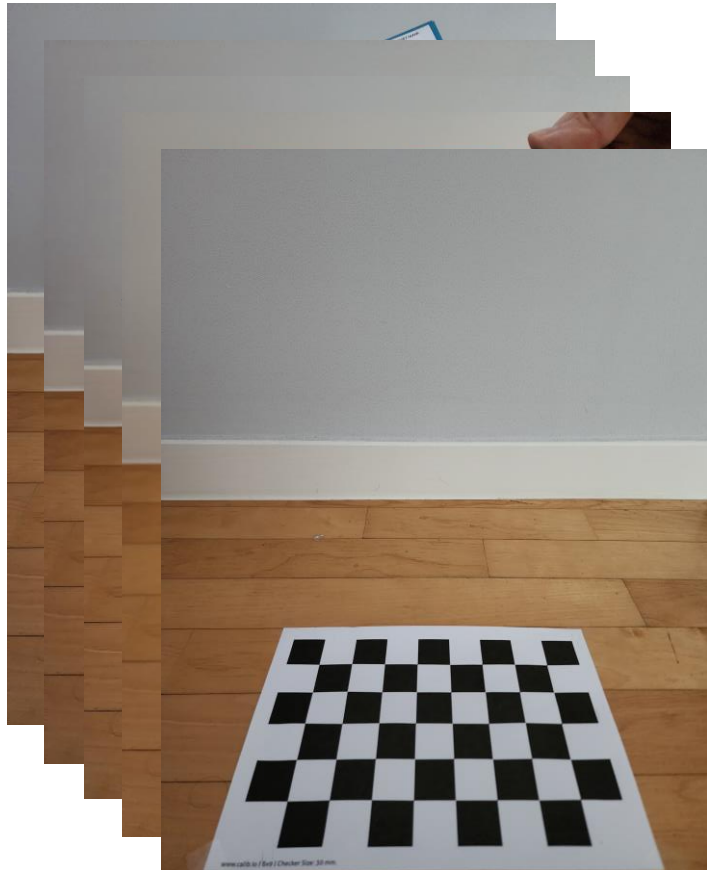
요소	설명
Intrinsic 행렬	카메라의 초점 거리 및 주점 정보를 가진 행렬
왜곡 계수	렌즈와 이미지간 정렬되지 않아 발생하는 왜곡
Rotation 벡터	이미지상의 좌표를 실제 좌표로 옮기는 회전 벡터
Translation 벡터	이미지상의 좌표를 실제 좌표로 옮기는 이동 벡터
World 좌표계	카메라가 인지하는 실제 세계의 좌표계



https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

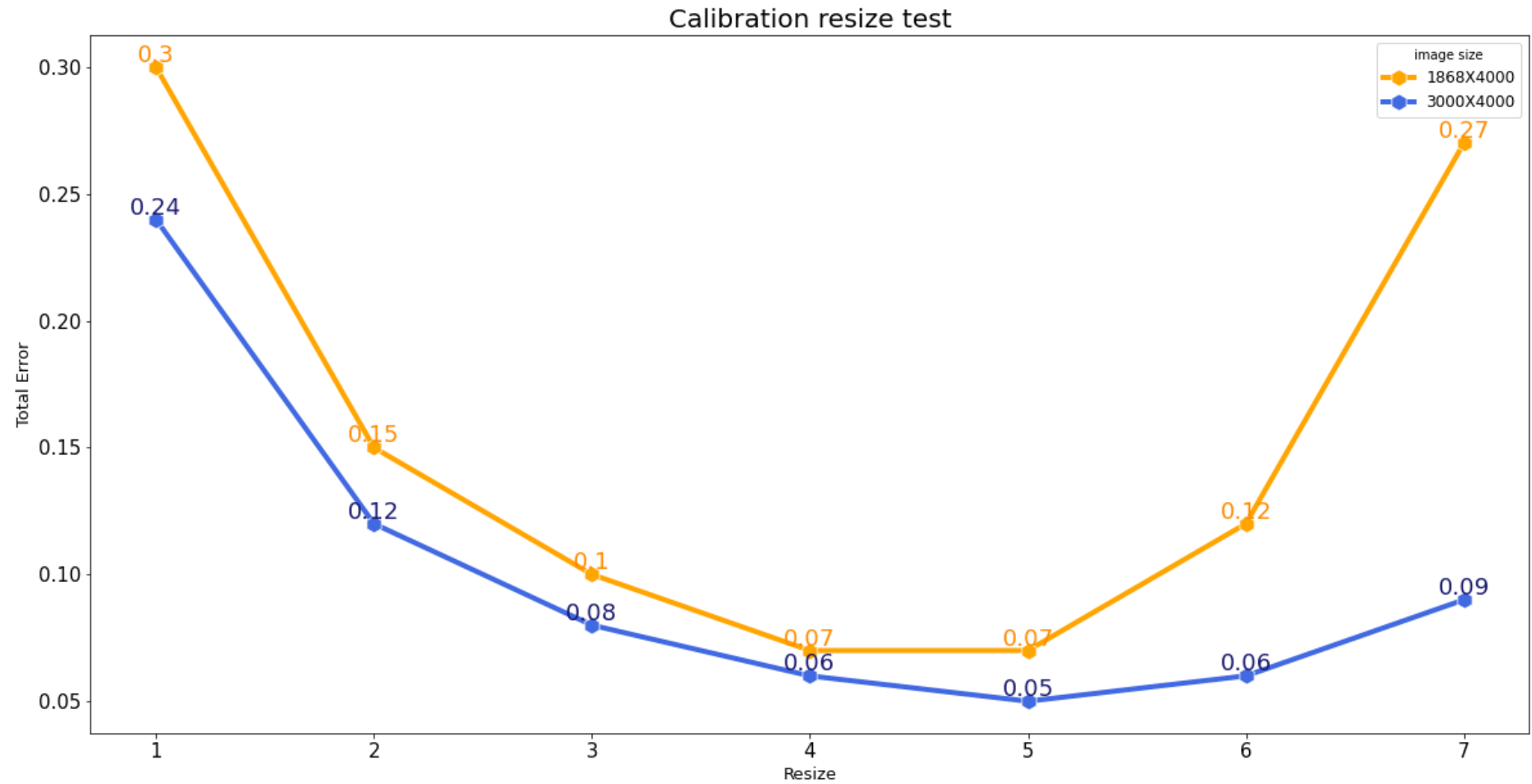
카메라 보정(Calibration)

- 다양한 관점에서 체커보드를 촬영하여 보정 시행
- 보정 시 처음으로 발견한 코너를 World 좌표계의 원점으로 인식



2.2

카메라 보정(Calibration)



- ✓ 변형과 코너를 찾는 알고리즘 간의 Absolute Norm을 계산결과
- ✓ 모든 Calibration 이미지에 대해 에러의 산술 평균

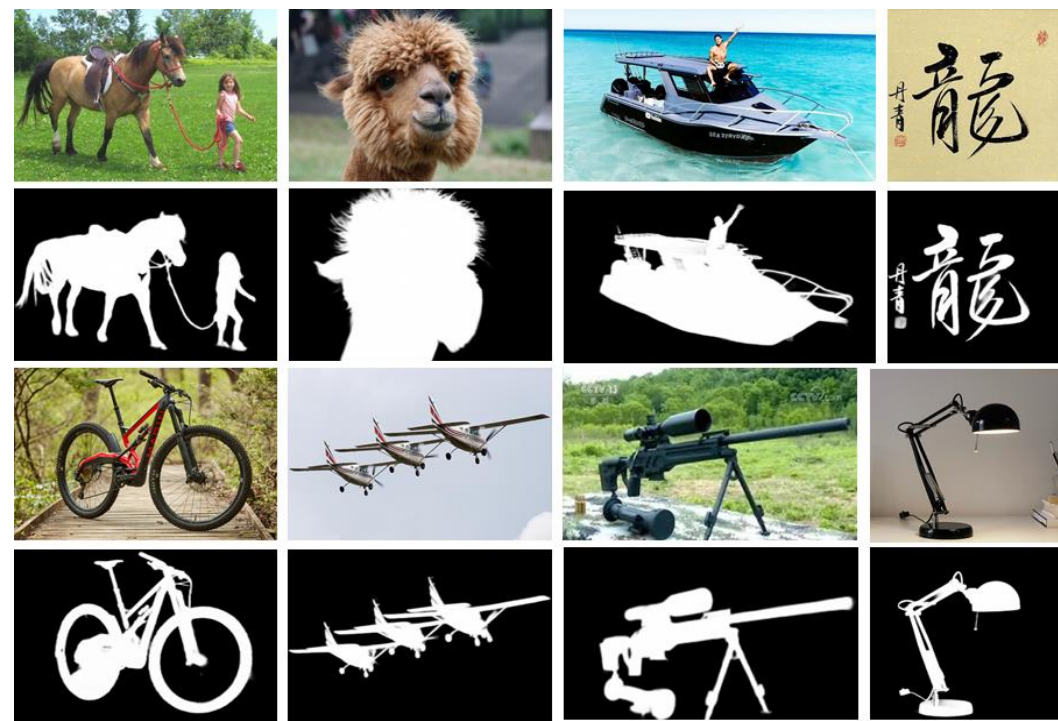
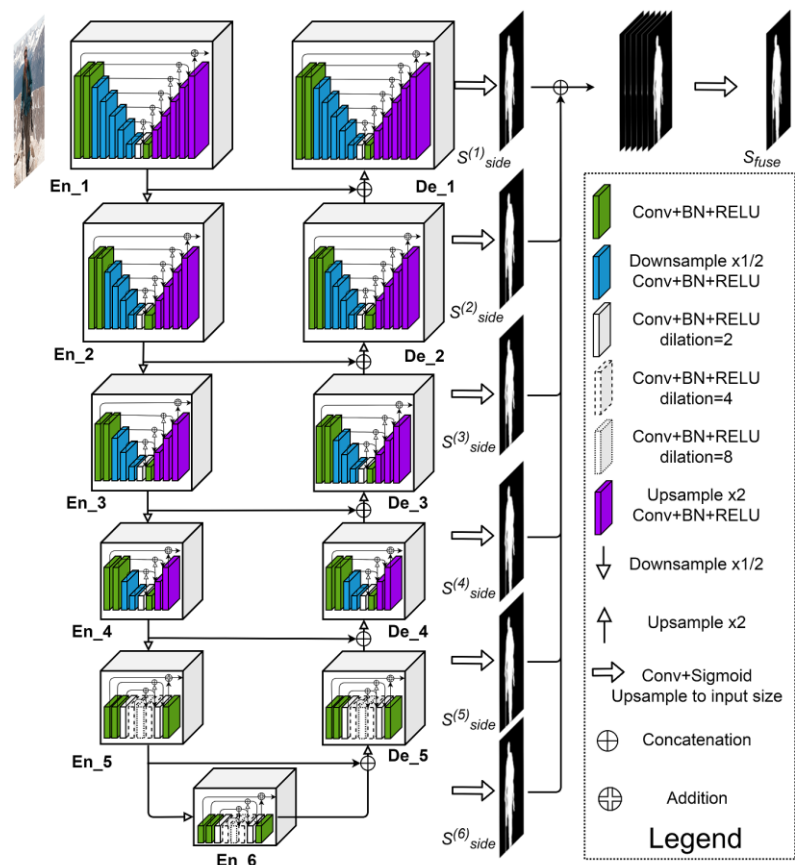
2.3 배경 제거

- 배경제거에는 U2-Net(U-Square Net) 기반의 Semantic Segmentation을 통해 이미지에서 물체에 해당하는 부분만 얻어냄



U2-Net(U-Square Net)

- U-Net이 있는 한 블록 = RSU(Residual U-block), RSU를 이용하여 모델을 구성.
- 각 side별로 나온 출력에 컨볼루션 연산, 시그모이드 연산, Up-Sampling을 진행.
- 모든 side의 출력들에 대해 Concatenation을 거쳐 최종 Segmentation 출력 반환.



출처 <<https://github.com/xuebinqin/U-2-Net>>

2.4 객체 탐지 (Object Detection)

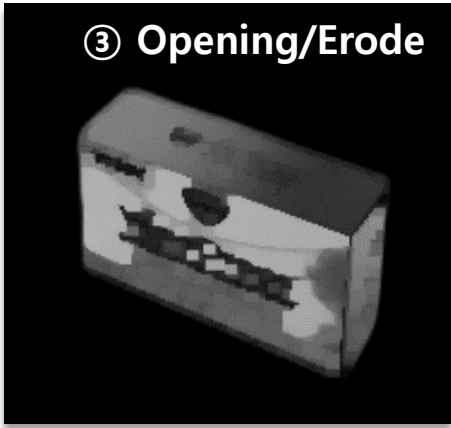
객체 탐지(Object Detection)



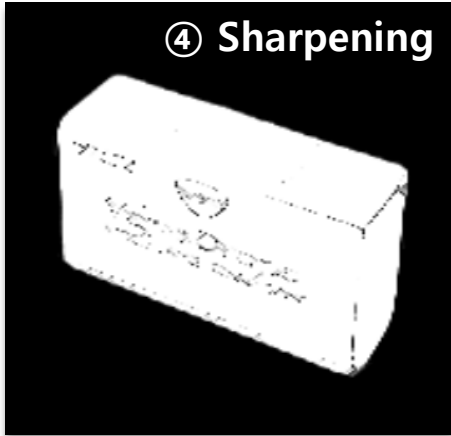
① 원본



② 배경제거



③ Opening/Erode



④ Sharpening

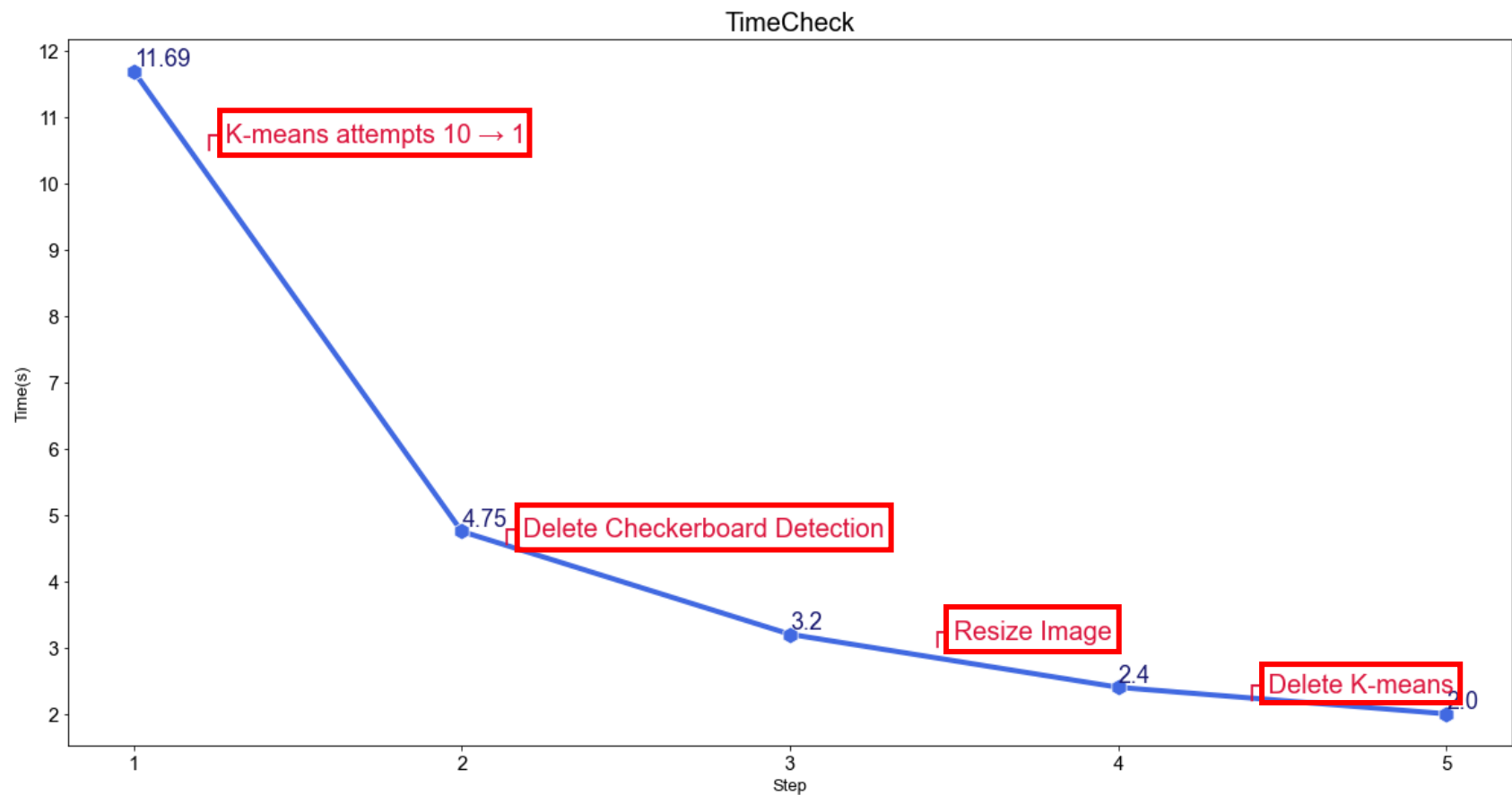
① 원본이미지

② 배경제거
물체와 배경 분리

③ Opening/Erode
이미지 팽창과 침식을 이용한
물체 경계 구분 명확

④ Sharpening
정확한 외곽선 검출을 위해
물체의 안쪽 텍스트를 삭제

객체 탐지(Object Detection)



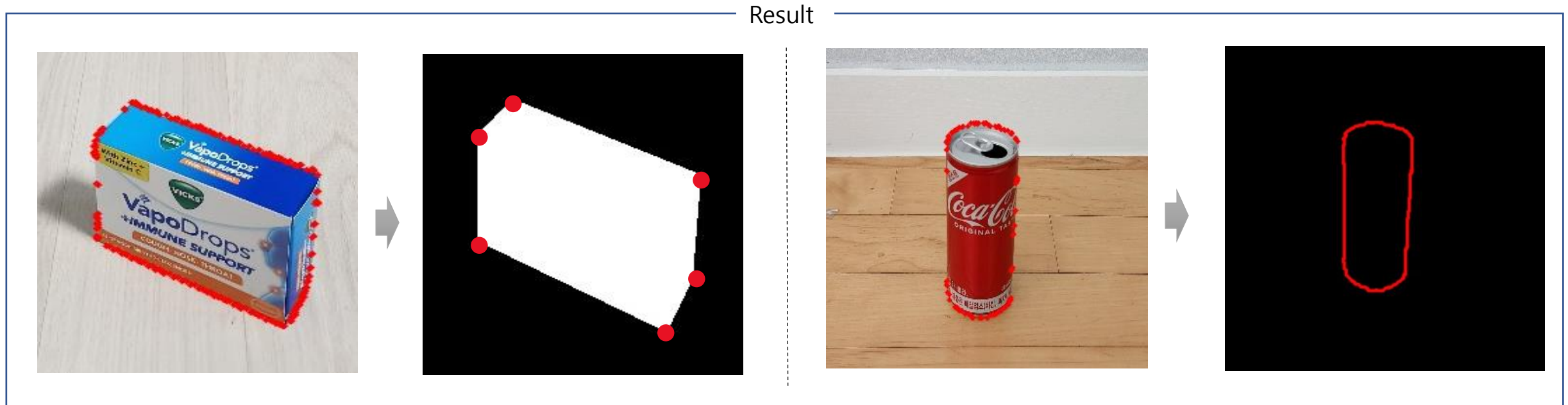
11.69초 → 2.0초

9.69초 단축!

2.5 외곽선 검출 (*findContours*)

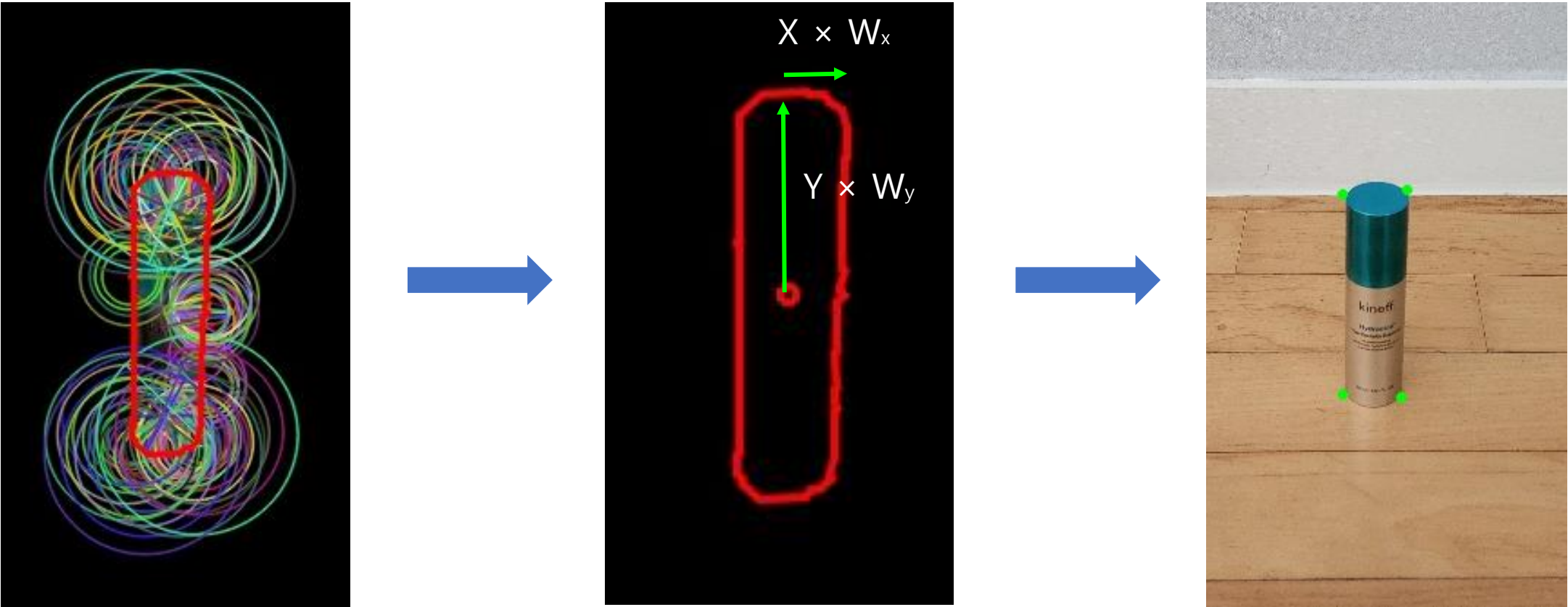
외곽선 검출(findContours)

- Object Detection이 완료된 이미지로 외곽선을 찾는 단계
 - 이미지에서 물체의 외곽선 좌표값들을 추출
 - 추출된 외곽선을 근사화하여 꼭짓점 좌표값들을 반환 (직육면체는 6개, 원기둥은 10~20개 사이의 꼭지점 좌표)
 - 추출한 외곽선 좌표값들을 통해 물체의 전체적인 외곽선 그리기



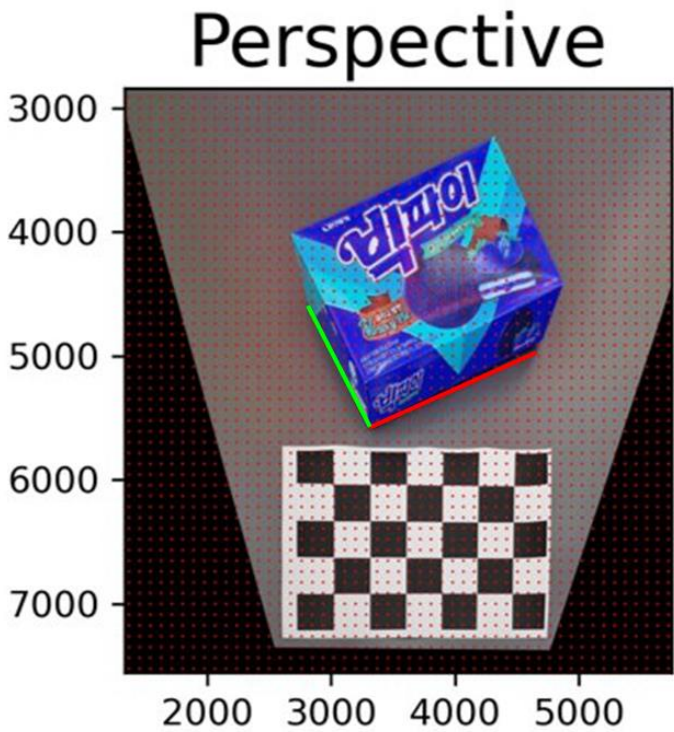
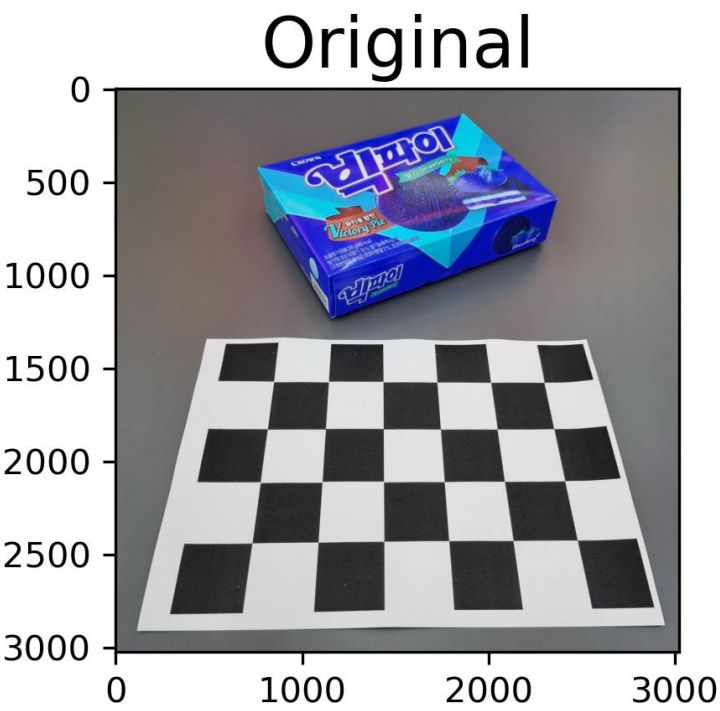
꼭짓점 검출(ORB)

- 체적 측정을 위한 꼭짓점 좌표를 구하기 위해 ORB를 사용
 - 외곽선의 중심점 좌표를 구하고 맨하탄 거리를 측정하여 가장 먼 거리의 꼭짓점들을 구함
 - x축, y축에 가중치 W 를 주어, x축에 조금 더 민감한 꼭짓점 검출



2.6 체적 도출

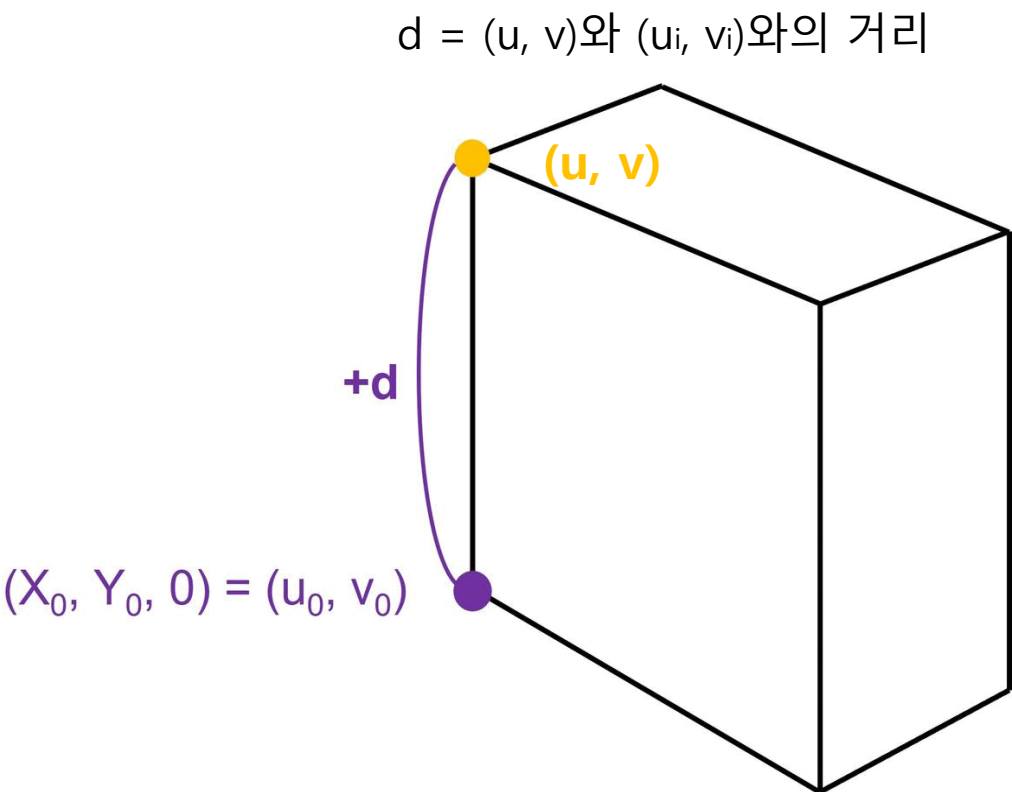
- Perspective Transform을 이용한 가로 세로 측정
Top view로 pixel 간의 간격당 실제 거리 측정 가능



$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

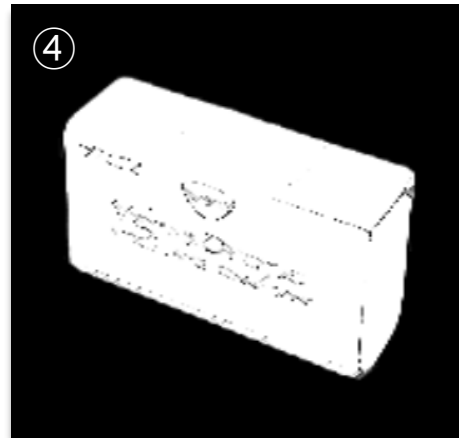
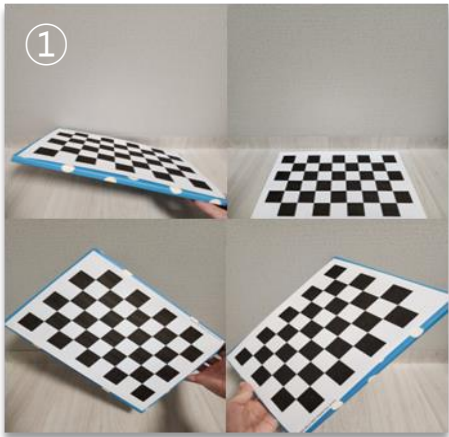
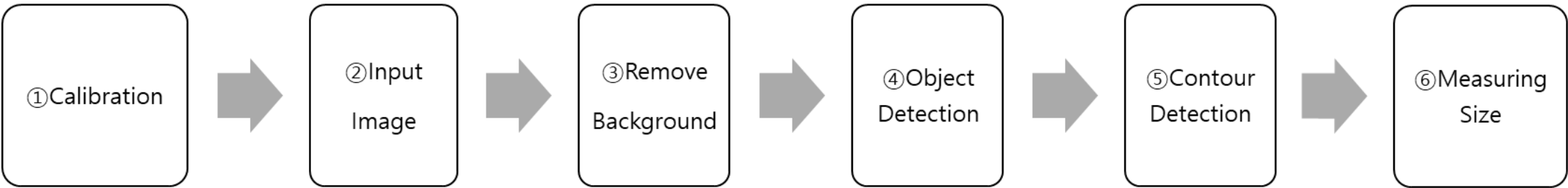
수식 1. 픽셀 좌표와 World 좌표 간의 관계식

- (u, v) : 이미지상의 픽셀 좌표
- f_x, f_y : 초점 거리
- c_x, c_y : 주점
- r : Rotation 행렬 요소
- t : Translation 벡터 요소
- (X_w, Y_w, Z_w) : World 좌표

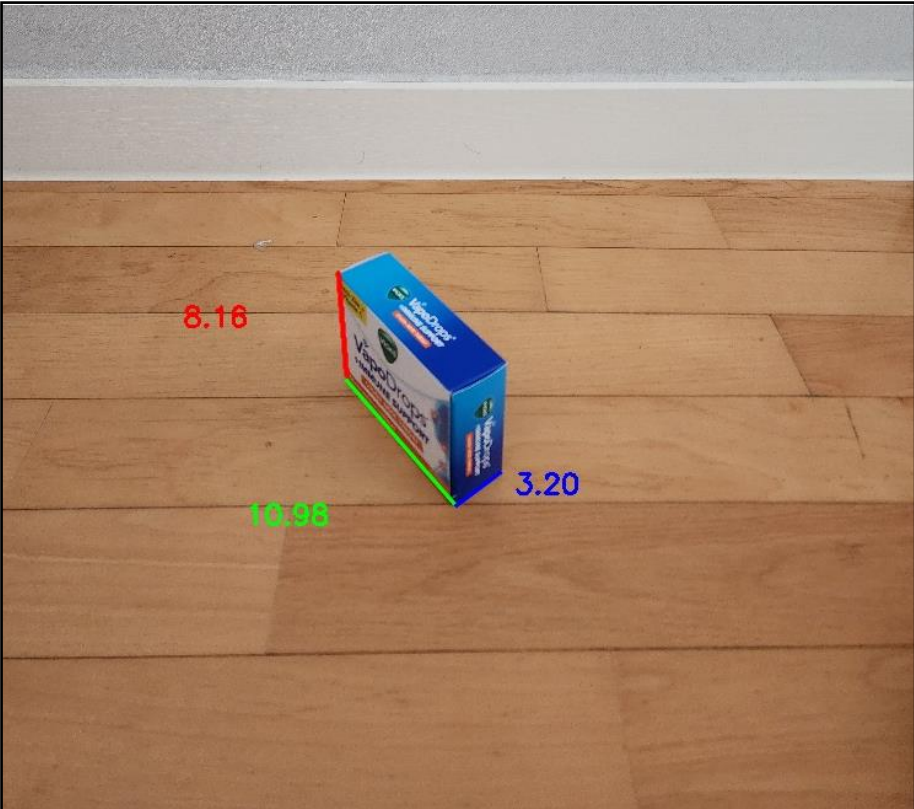


3. 실측 테스트 결과

실행 순서



실행 결과(직육면체)



	가로	세로	높이
실제사이즈	11.10	3.50	8.00
측정사이즈	10.98	3.20	8.18

단위: [cm]



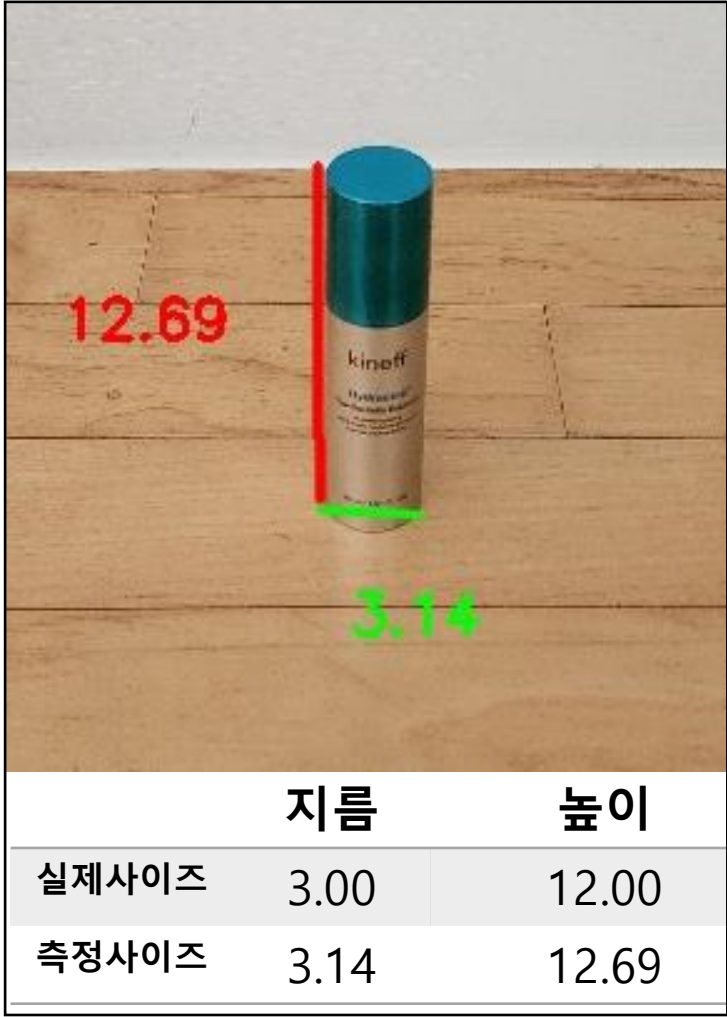
	가로	세로	높이
실제사이즈	14.50	7.70	7.30
측정사이즈	14.72	7.57	7.05

단위: [cm]

실행 결과(원기둥)



단위: [cm]



단위: [cm]

3.1

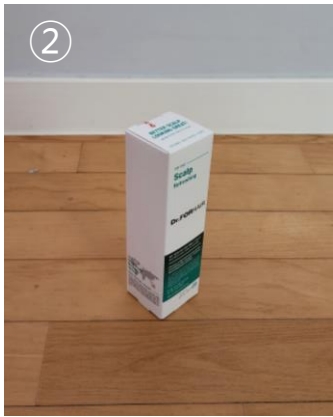
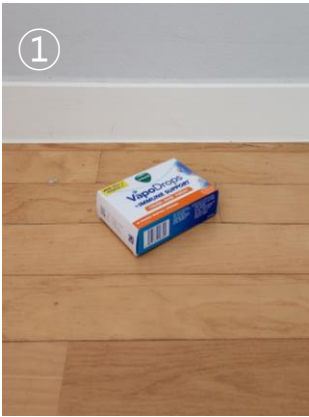
정량평가 결과(직육면체)

단위: [cm]

	실 제			오 차			평균
	가로	세로	높이	가로	세로	높이	
① 감기약	3.50	11.10	8.00	0.09	0.24	0.19	0.15
② 스프레이	4.50	18.00	4.50	0.00	0.07	0.09	0.00
③ 콧부차	4.40	14.00	8.50	0.20	0.30	0.14	0.19
④ 여주분말	7.30	7.70	14.50	0.02	0.27	0.26	0.05
⑤ 미숫가루	18.40	17.00	14.00	0.47	0.96	0.08	0.19
⑥ 종이박스	21.00	21.00	29.00	0.19	0.12	0.05	0.09

오차율 평균

→ 2.276%



3.1

정량평가 결과(원기둥)

단위: [cm]

	실 제		오 차		평균
	지름	높이	지름	높이	
① 장조림	7.30	3.20	0.07	1.27	0.13
② 평창жат	7.00	9.10	0.61	1.26	0.82
③ 핸드크림	6.00	6.50	0.16	0.55	0.25
④ 클리너	8.30	6.50	0.04	1.21	0.08
⑤ 화장품	3.00	12.30	0.14	0.69	0.23
⑥ 코카콜라	4.90	13.30	0.19	0.62	0.29
⑦ Rice	10.50	14.00	0.32	1.03	0.49

오차율 평균

→ 6.752%





프로젝트를 진행해보니

- 웹(Flask) 및 앱(Flutter)으로 구현



Flask



- 굴곡 있는 원기둥



Q&A

부록

부록1. 기획 과정 변경 사항



그림 1. 물체가 올라가 있는 체커보드.



그림 2. 물체와 분리된 체커보드.



그림 3. 체커보드 제거 후 물체

1-Step

- 체커보드 위에 물체를 올려놓은 상태로 촬영
- 체커보드로 인해 물체를 인지하지 못하는 단점이 존재

2-Step

- 체커보드와 물체를 분리
- 체커보드가 물체 인식에 영향을 미치지 않음
- 체커보드보다 큰 사물의 크기를 측정할 수 있음

3-Step

- 체커보드 제거
- 체커보드 인식 시간 단축
- 카메라 위치 고정

부록2. Object Detection Preprocessing

물체의 외곽선을 찾기 전에 물체의 외곽선을 좀 더 정확하게 찾기 위해 진행되는 과정.

- MorphologyEX - Binary이미지에만 적용가능
 - Opening : 침식(erosion) 후에 팽창(dilation)을 진행하는 연산으로 이미지상의 노이즈제거에 사용.



노이즈 있는 이미지



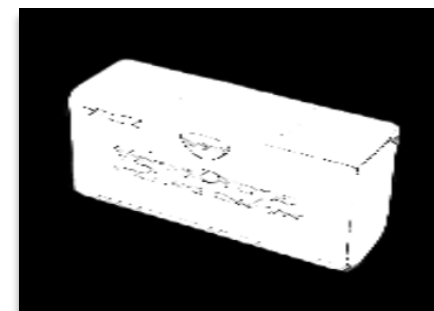
Opening 결과

- Sharpening

Sharpening 기법이란 마스크를 주변의 값들과 대비되도록 만든다.

Sharpening 원래의 화소 값을 n 배, 주변의 화소 값을 -1 배 하여 합이 1이 되도록 마스크를 만든다.

배경제거 후 물체 주변이 검은 바탕이기 때문에 Sharpening의 일정 이상의 값을 주면 물체의 안쪽이 전체적으로 하얗게 처리된다. (안쪽 글씨를 잡는경향)



출처 <<https://webnautes.tistory.com/m/1257>>

부록3. K-means

k-means 알고리즘을 이용한 컬러 영상 분할

색상 양자화라는 용어를 쓰기도 합니다.

영상에 존재하는 1600만 가지의 **컬러값을 10개(k, 군집 개수)로 단순화**할 수 있습니다.

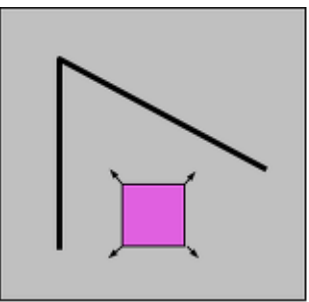
색 공간에서 k-means 알고리즘을 수행하고 각 픽셀 값을 k개의 대표 색상으로 치환하게 됩니다.



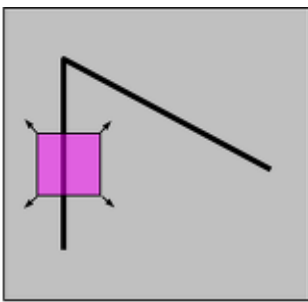
부록4. **꼭짓점 검출(ORB)**

영상에서 코너(corner)를 찾는 기본적인 아이디어는 아래 그림과 같이 영상에서 작은 윈도우를 조금씩 이동(shift)시켰을 때, 코너점의 경우는 모든 방향으로 영상변화가 커야 한다는 점입니다.

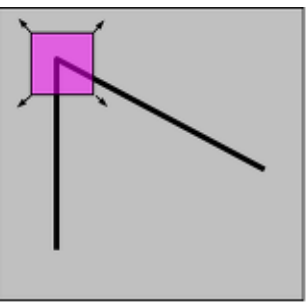
ORB (Oriented and Rotated BRIEF)
디스크립터 검출기 중 BRIEF(Binary Robust Independent Elementary Features)라는 것이 있습니다. BRIEF는 특징점 검출은 지원하지 않는 디스크립터 추출기입니다. 이 BRIEF에 방향과 회전을 고려하도록 개선한 알고리즘이 바로 ORB입니다. 이 알고리즘은 특징점 검출 알고리즘으로 [FAST](#)를 사용하고 회전과 방향을 고려하도록 개선했으며 속도도 빨라 SIFT와 SURF의 좋은 대안으로 사용됩니다. ORB 객체 생성은 다음과 같이 합니다.



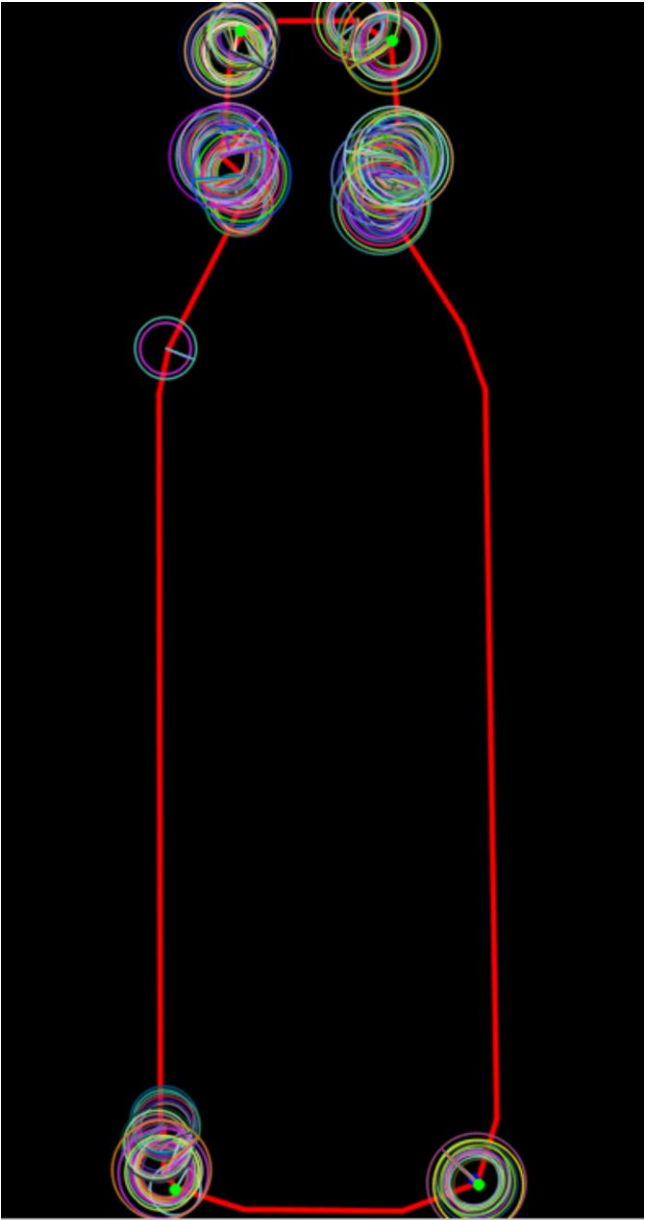
“flat” region:
no change in all directions



“edge”:
no change along the edge direction



“corner”:
significant change in all directions

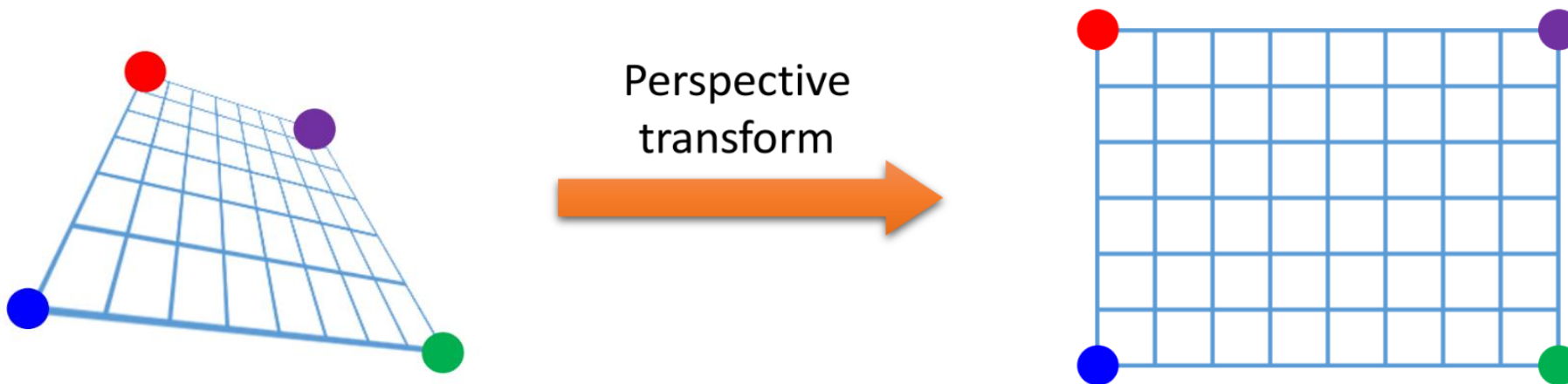


출처 <<https://bkshin.tistory.com/m/entry/OpenCV-27-이미지의특징과키포인트>>

부록5. Perspective Transform

Perspective Transform(원근 변환)

보는 사람의 시각에 따라 같은 물체도 먼 것은 작게, 가까운 것은 크게 보이는 현상인 원근감을 주는 변환. 이미지상에서 원근에 따라 줄어든 물체의 픽셀좌표를 입력해주어 플랫폼한 상태로 새로운 좌표값을 반환.

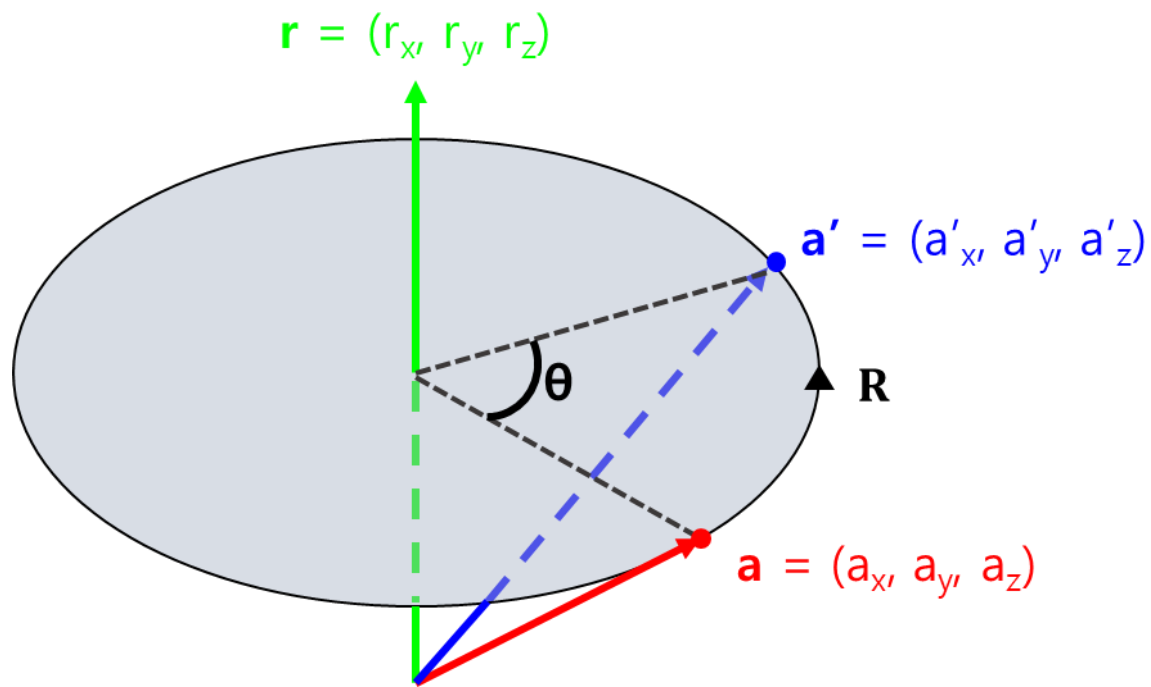


3차원에서의 물체를 2차원 평면이미지로 투영하는것.

이 때 사용되는 변환행렬을 사용해서 물체를 평면으로 만들었다 다시 3차원으로 되돌릴 수도 있음.

부록6. Rodrigues' Rotation

- Calibration 시 회전에 대한 정보는 회전축 벡터로 저장
- Rodrigues 변환을 통해 회전 행렬로 반환



$$\theta = \sqrt{r_x^2 + r_y^2 + r_z^2}, \quad \hat{\mathbf{r}} = (\hat{r}_x, \hat{r}_y, \hat{r}_z), \quad K = \begin{pmatrix} 0 & -\hat{r}_z & \hat{r}_y \\ \hat{r}_z & 0 & -\hat{r}_x \\ -\hat{r}_y & \hat{r}_x & 0 \end{pmatrix}$$
$$\mathbf{a}' = (\cos\theta I + \sin\theta K + (1 - \cos\theta)\hat{\mathbf{r}}\hat{\mathbf{r}}^T)\mathbf{a}$$
$$\therefore \mathbf{a}' = R\mathbf{a}$$