RGBD image 를 이용한 발의 특성 측정

<1 조>: 윤영식 (19101381), 오상민 (17101323)
서울과학기술대학교 <전기정보공학과>

요 약

Color image 와 그에 대응되는 depth image 를 통해 발의 3D point cloud 를 생성하고 발의 특성을 분석한다. 분석할 발의 특성은 발의 길이, 발의 폭, 발의 높이이다. 발의 사진은 kinect camera 로 찍으며, 이때 맨발을 A4 위에 올려놓고 찍는다. 일련의 과정은 특정한 상황에서 자동으로 수행되고 그렇지 않으면 사용자의 입력을 받아서 수행해야 된다.

키워드: 맨발, Kinect camera, A4, 2D segment, point cloud, 3d rigid transform

1. 서론

전자 상거래, 물류 기술 등의 발전으로 온라인 쇼핑이 늘어가는 추세이다. 온라인 쇼핑몰 취급상품범위별/상품군별 거래액(통계청)에 따르면 2017 년에는 94 조, 2018 년에는 113 조, 2019 년에는 136 조, 2020 년에는 159 조로 꾸준히 증가함을 볼 수 있다.[i]

이에 맞춰 의류/신발과 같은 상품군의 거래액도 증가하는데, 오프라인매장에서 사이즈를 맞추어 보고 사는 경우가 일반적인 위 상품군을 온라인에서 구매할 시, 의류의 경우 총장, 어깨너비, 허리 단면 등의 정보를 활용하여 사이즈를 예측하는데 비해, 신발의 경우 발의 길이만으로 사이즈를 예측하기에 어려움이 있다.

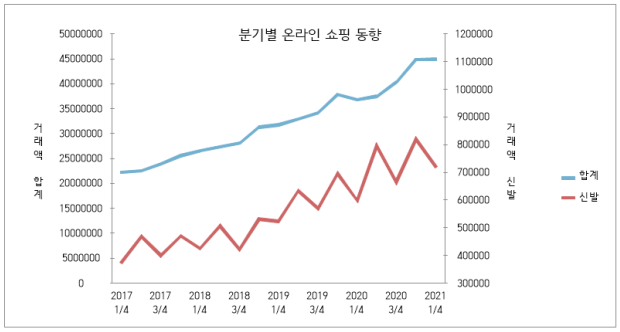


그림 1 . 분기별 온라인 쇼핑 동향(단위: 백만원)

이는 삼차원 체적을 일차원 길이로만 측정, 일반적인 발의 모델과 비교하기 때문에 일어나는 문제이다. 실제 사람의 발은 그 모양이 상이하므로 이를 해결하기 위해선 2 차원의 길이, 너비 정보를 사용하거나 더 나아가 3 차원의 높이정보까지 이용함으로써 정확한 발의 특성을 얻을 수 있을 것이다.

현재 국내에서 발 사진의 촬영으로 발 형태를 분석한 선행 연구로 Perfitt 사의 신발 추천 알고리즘이 있다.[ii] 발과 수직으로 촬영한 사진을 딥러닝 알고리즘으로 발의 2 차원 정보인 길이, 너비를 측정하고 적절한 신발을 추천하는 서비스이다.

이와 같이 사진을 이용하여 발의 모델링을 하고자 하는 시도가 늘어남에 있어 본 실험은 3D TOF 카메라를 사용, 한 번의 촬영으로 발의 PCD 를 만들어 정보인 길이, 너비, 높이를 측정하는데 그 목적이 있다.

또한 우리는 사진 한 장을 사용하여 손실되는 발의 정보들이 있었으나 여러 장을 찍어 matching 하여 발의 모든 정보를 얻을 수 있을 것으로 보인다.

본 실험 결과를 활용하면 보다 간단하게 발의 모델링이 가능해지며, 이는 가상으로 신발을 신기는 등의 확장이 가능할 것이다.[1]

그림 2. 입력과 출력결과

[1] CloTH-VTON plus : Clothing Three-Dimensional Reconstruction for Hybrid Image-Based Virtual Try-ON, IEEE ACCESS, vol.9 pp.30960~30978, 2021 참고

1.1 변경 사항

표 1. 변경내용

	변경내용	사유
1	Grab cut 미사용	Grab cut 이 좋은 결과를 얻으려면 사용자와 interaction 이 있어야 하지만 자동으로 하려는 프로젝트 목적상 부합하지 않았다.
2	Perspective transform 미사용	Perspective transform 은 2D 에서 카메라의 시점을 바꾸게 되어 3d 로 변환하게 되면 scale 에 문제가 생기거나 shape 에 문제가 생겼다.

표 1.2 과제를 하면서 변경된 사항

	변경내용	실패 이유
1	Opencv Hough line	본래 hough line 의 소실점을 구해 a4 용지의 꼭짓점을 검출 하려했으나 카메라 자체의 왜곡 때문에 a4 용지의 모서리가 완전한 직선이 아니게 되었다.
2	렌즈 왜곡 보정	위의 허프라인 문제를 해결하기 위해 렌즈의 왜곡 파라미터를 수동으로 찾아 왜곡을 보정해주려 했다. 하지만 센서 별 왜곡 파라미터의 차이, scale 문제가 발생하였다. 또한 color image 와 depth image 의 왜곡이 각각 달라서 보정하기 힘들다.

2. 방법 (알고리즘)

핵심은 발 영역을 자동으로 분할하는 것이다. 발을 정확하게 분할해야 정확한 발 영역 point cloud 를 얻을 수 있고, 그래야 정확한 발의 특성을 얻을 수 있다.

다음은 실험 환경이다.

1. A4 위 발을 얹은 뒤 촬영

   A4 를 사용한 이유는 다음과 같다.

   첫째, Kinect camera 로 얻어진 ply 파일을 분석해본 결과 실제 측정 길이와 차이가 있었다. 이러한 scale 문제를 보정해주기 위해 210 x 297 mm 으로 정형화된 크기가 보장되는 A4 를 사용하였다.

   둘째, 또한 밝기가 밝고 균일한 A4 를 사용하면 바닥과 발, A4 영역 분할에 있어 이점을 가지기 때문에 A4 를 사용하였다.

2. 발측정의 척도

   발의 길이와 폭은 발이 axis 에 aligned 됐을 때의 bounding box 의 세로 길이와 가로 길이로 잡았다.

   발의 높이는 발바닥에서 발목까지의 거리로 잡았다. 발목은 발등과 다리의 경계 즉 발을 기울이지 않았을 때 높이가 급격하게 변하는 지점이다.

   이를 구하기 위해 axis aligned 된 발의 point cloud 를 yz 평면에 projection 했을 때 z 간격 10mm 의 범위 안 point 의 개수와 범위 안 point 간 최대 y 거리를 구한다. 구간을 나눈 이유는 사진 한 장만 사용하므로 point 의 개수가 부족하기 때문이다. point 개수가 max point 개수의 0.1 배인 구간은 확실하게 다리인 부분이므로 후보군에서 제외가 되며, 통상적으로 발목은 50mm 이상이므로 그 이하 구간 역시 제외한다. 그 후 z 의 gradient 에 대한 히스토그램을 그렸을 때 maximum 구간이 발목의 높이 이다

   .

3. 발 측정 환경

   발의 측정을 자동으로 하기 위해 촬영 환경을 다음과 같이 설정하였다. 1 에서 설명했던 이유로 A4 위에 발을 올려 놓고 사진을 찍어야 한다. 이때 A4 의 4 꼭짓점이 모두 보여야 하고, 발이 A4 의 연결되는 모서리 2 개 이상을 가리거나 너무 인접해서는 안 된다. 이때 바닥의 반사광에 의해 A4 의 꼭짓점이 가려지는 경우 역시 불가능하다. 이러한 경우를 충족하지 못하면 결과가 부정확하게 나온다. 이 경우 마우스 이벤트로 사용자가 A4 의 꼭짓점을 구하도록 구현하였다. 허프 변환은 1.2 와 같은 이유로 사용하지 못하고 corner detect 를 사용하게 되었다.

   또한 히스토그램 역투영을 사용하여 발을 검출하기 위해 맨발로 사진을 찍는다.

   실험에서 10 명의 발 데이터를 찍었는데, SOTA 와 같은 환경에서 비교하기 위해 발뒤꿈치가 a4 모서리에 닿도록 하고 정면에서 찍은 사진을 사용하여 비교하였다.

그림 3. 발 촬영 예시(SOTA 와 같은 환경)

또한 우리는 발 뒷꿈치를 a4 모서리에 닿지 않더라도 발의 특성을 측정할 수 있도록 다음과 같이 발을 a4 의 중앙에 놓고 발 뒷꿈치가 보이도록 다리를 a4 의 왼쪽 혹은 오른쪽 모서리를 가리도록 놓는 방식으로도 측정하였다. 이러한 방식은 SOTA 의 방식보다 개선되었다고 본다. SOTA 의 방식으로 찍었을 때랑 다음 방식대로 찍었을 때 값의 차이 또한 실측정값과 비교할 것이다.

그림 4. 발 촬영 예시 2 (SOTA 와 다른 각도)

2.1 2D segment

첫 번째로 바닥과 발을 포함한 a4 를 분리한다. 이 이미지는 scale 을 보정하는데 쓰인다. 두 번째로 a4 와 발을 각각 분리한다. A4 는 평면의 방정식을 구하는데 사용된다. 발은 발의 특성을 구하는데 사용된다.

이를 위해 Watershed 알고리즘으로 a4 를 segment 하고 그것의 convex hull 을 구하여 a4 와 발이 있는 영역을 구한다. 이러한 연산은 depth 이미지와 color 이미지 모두 같게 적용되어 3D 상에서 point 값, 즉 좌표 값과 그 점의 color 값이 대응하도록 한다.

2.1.1 sure a4 detect

watershed 를 적용하기 위해 확실하게 a4 인 부분을 찾을 것이다. 명도가 높고, 그 값이 거의 일정한 a4 용지의 특징을 사용하여 확실하게 a4 인 부분을 찾을 것이다. 본래 a4 가 사각형이라는 기하적인 특성도 사용하려 했으나 카메라 렌즈 왜곡으로 인해 사각형으로 인식하지 않는 문제가 있었다.

위의 특성을 모두 사용하기 위해 적당한 threshold 값을 잡고 threshold 를 grad 이미지와 gray 이미지에 적용하고 이를 bitwise_and 한 mask 를 구한다. 그리고 하나의 contour 만 남도록 연산한다.

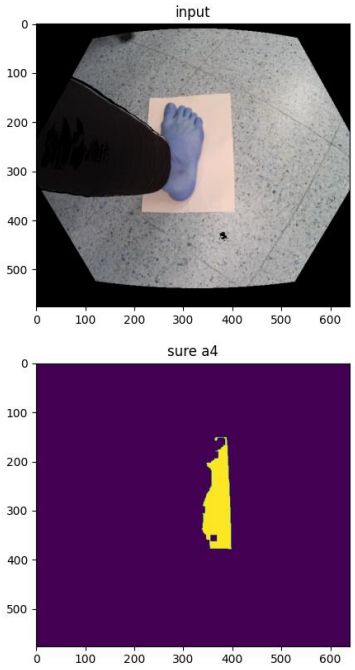


그림 5. Input 이미지에 따른 sure a4 검출

2.1.2 sure not a4 detect

확실하게 a4 가 아닌 부분은 바닥, 다리(발) 부분이다. 다리와 바닥은 반드시 이미지의 최외곽에 있을 것이고, 발은 맨발이므로 histogram backpropagation 으로 구한다.

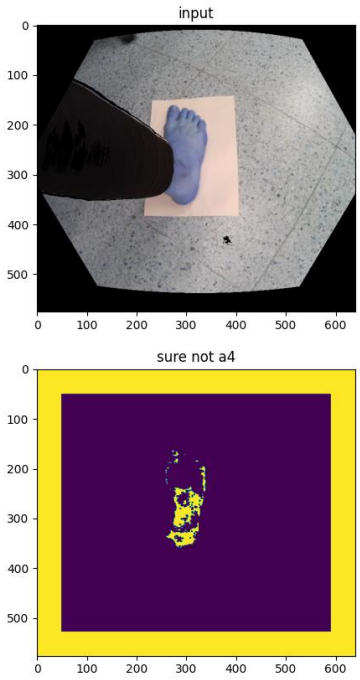


그림 6. Input 이미지에 따른 sure not a4 검출

2.1.3 watershed segmentation

1 과 2 의 결과를 이용하여 watershed 를 적용한다. 이때 flooding 이 발생할 수 있으므로 이미지의 gradient 의 threshold_inv 값을 bitwise 연산한 후 morphological filter 로 잡음처럼 나온 바닥 부분을 최대한 정리한다. 그 이후 가장 길이가 긴 contour 에 대한 convexhull 을 구한다. 이는 a4 의 명도 차가 크지 않다는 특성을 이용한 것이다. (발 부분도 손상이 간 이유는 발을 a4 에 놓았을 때 a4 가 찌그러졌기 때문이다)

다음은 flooding 이 발생했을 때의 대처 과정이다.

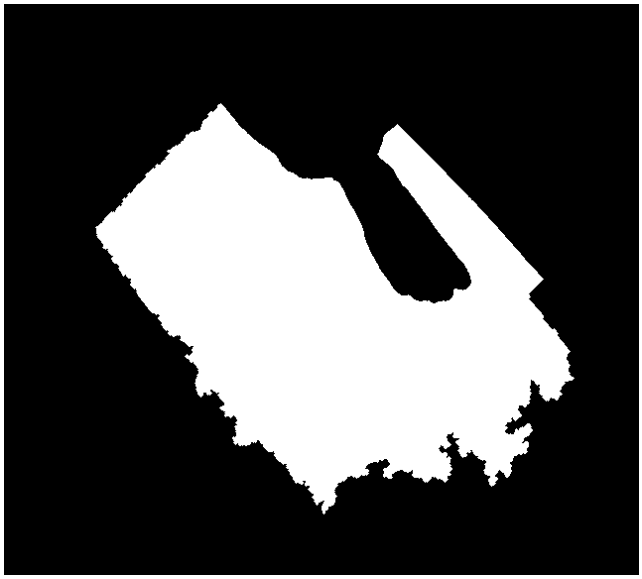


그림 7. Flooding 이 발생한 경우

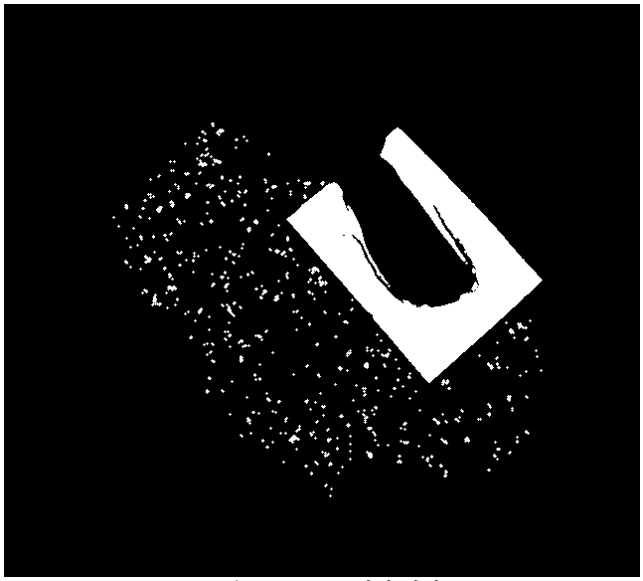


그림 8. Flooding 대처 결과

2.1.4 a4 와 발의 segmentation

3 의 convex hull 을 구하면 a4 를 둘러싸는 사각형에 가까운 도형을 구할 수 있다. 이를 이용하면 발 또한 얻을 수 있다. 이를 histogram back propagation 과 watershed 로 a4 와 발로 분리할 수 있다.

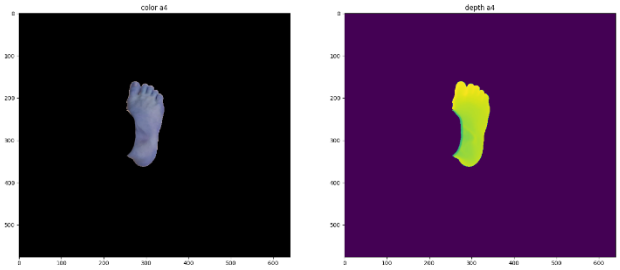


그림 9. 발 영역 분할

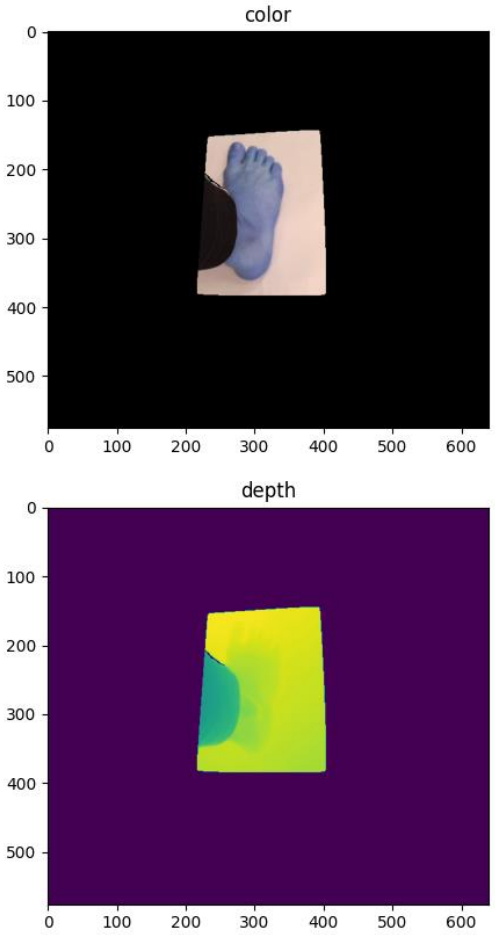


그림 10. a4 영역 분할

2.2 pcd transform

pcd 를 얻어 이를 회전, 이동시켜 xyz 좌표축에 정렬한다. (extrinsic 을 구한다)

2.2.1 2.1 에서 얻은 color 와 depth 이미지를 open3d 의 함수를 통해 pcd 로 변환한다. 이때 노이즈를 제거하기 위해 hidden point remove 함수를 사용한다.

그림 11. Hidden point 를 제거한 발 영역 pcd

2.2.2 Open3d 의 함수를 이용해 바닥부분 즉 a4 부분만 검출한다. 이 때 plane 의 방정식을 얻는다.

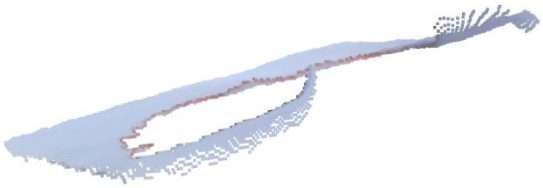


그림 12. A4 영역 pcd

2.2.3 Plane 의 방정식으로 절편을 구하고 이러한 3 점을 대응점으로 하여 3d rigid transform 을 적용하여 xy 평면에 평면이 있도록 한다. 이때 헤론의 공식을 통해 대응점의 좌표를 계산한다.

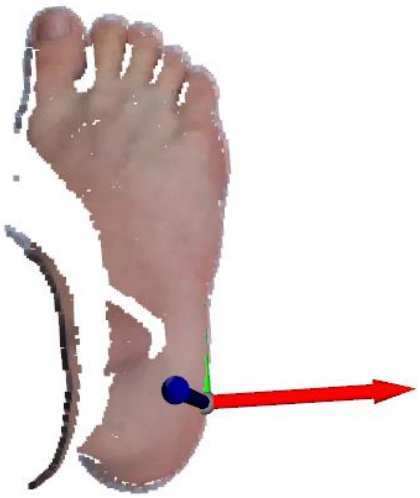


그림 13. xy 평면에 정렬한 발 영역 pcd

2.2.4 발의 pcd point 의 좌표값을 3 차원 Qhull 의 vertex 집합으로 변환하여 x,y 의 평균, 공분산 행렬을 얻는다. 그 후 특잇값 분해를 이용하여 eigenvector 를 얻은뒤 x,y 평균, eigenvector 를 반환한다. 이때 eigenvector 가 발의 장축이 되므로 이를 이용하여 발을 축에 맞출 수 있다.

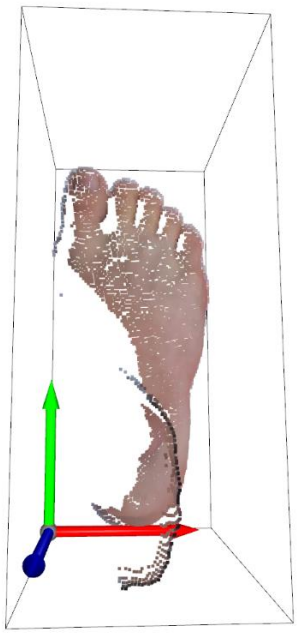


그림 14. xyz 축에 정렬시킨 발 영역 pcd

2.3 measure

2.3.1 축에 정렬된 발에 대한 axis aligned bounding box 를 구하여 발의 폭과 길이를 구한다. 이때 2.2.1 의 hidden point removal 을 사용해도 카메라의 calibration 문제로 발에 가려진 부분에 노이즈가 남는 경우가 있다. 따라서 폭을 구할 때 노이즈가 없는 부분만 자른 후 길이를 측정한다. 실험에서는 발의 길이의 절반을 기준으로 두 영역으로 나눈 후 노이즈가 발 뒤꿈치 쪽에 생기므로 발의 앞부분의 box 를 구해 폭을 구한다.

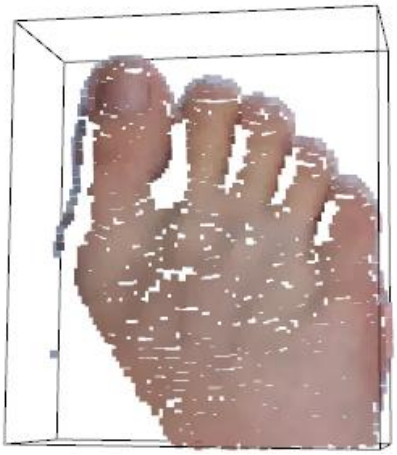


그림 15. 노이즈가 없는 앞부분을 잘라낸 발 영역 pcd

2.3.2 발의 높이 즉 바닥에서 발목까지의 거리를 측정하기 위해 발을 yz 평면에 projection 한다. 그 후 z 값의 범위(5mm 간격)에 따라 점의 개수 및 점간 최대 길이를 구하여 발목까지의 높이를 구한다. 범위를 나눈 이유는 나누지 않으면 사진 한 장으로 나오는 적은 수의 점들로는 발의 높이를 충분히 계산하기 힘들기 때문이다.
실험에서 발의 높이가 보통 50mm 이상이므로 그 아래 점들의 분포는 신경 쓰지 않았고 점의 개수가 최대의 0.05 도 안 된다면 그 부분은 확실하게 다리일 테니 그 영역 역시 무시한다. 발목은 발등과 다리의 경계이므로 점간 거리와 개수가 급격하게 변한다. 이를 이용해서 위의 영역을 무시한 영역에서 gradient 의 최대값을 구하면 그 때 높이가 발의 높이일 것이다.

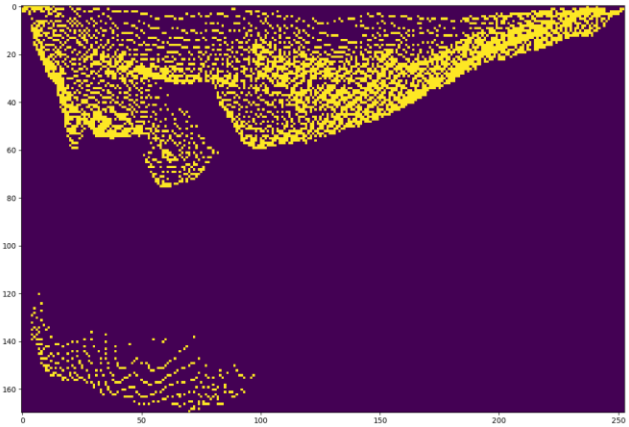


그림 15. yz 평면 투영 결과 1

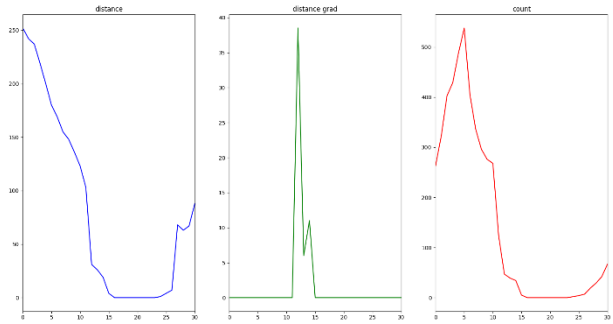


그림 16. yz 평면 투영 분석

2.3.3 위의 이미지로 pcd 를 변환하였을 때 a4 길이가 실제 길이 210x297mm 와 안 맞는 경우가 있었다. 따라서 이를 보정해주기 위해 pcd 상에서 a4 의 길이를 구한다. 이를 구하기 위해 a4 의 pcd 를 구하고 이를 xy 평면과 평행하게 transform 하고 점들의 convex hull 을 구해 꼭짓점을 구한다. 그리고 그 점을 통해 회전하여 a4 의 axis aligned bounding box 를 구하고 이를 통해 길이를 측정하고 보정한다.
마우스 이벤트로 가는 경우 a4 의 정보가 가려져 있으므로 위의 scale 보정을 하면 문제가 생긴다.

2.3 mouse event

아래의 사진처럼 a4 가 제대로 찍히지 않거나 바닥의 빛 반사로 인하여 a4 의 정보가 손실되는 경우, 혹은 바닥이 a4 와 비슷하여 segment 가 제대로 작동되지 않는 경우가 있다. 이 경우 재촬영 하는 대신 사용자가 a4 의 4 꼭짓점을 지정하여 해결한다.

그림 17. A4 의 정보가 손실된 경우

그림 18. 바닥이 A4 와 유사한 경우

그림 19. 마우스 이벤트 적용

3. 구 현

구현된 코드를 포함한 시스템을 블록 다이어그램과 계층 형태로 그렸으나. 그림 사이즈가 커 붙임에 게시하였다.

다음은 활용한 공개 라이브러리의 주요 API 와 개발한 코드 내용을 표로 정리한 것이다.

표 2. 활용한 공개 라이브러리 주요 API [iii] [iv] [v]

함수	설명
Cv2. watershed	Color image 랑 seed 가 그려진 marker 를 입력하면 그 seed 가 확장되어서 경계(edge)에서 멈춰, 영역이 된다. 경계면이 뚜렷하고 seed 를 잘 marking 하면 영역분할 하는데 효과적이다.
Cv2. convexhull	Contour points 를 모두 포함하는 볼록한 외곽선을 그린다. A4 만 segment 하면 오목한 부분이 있어 convexhull 로 직사각형에 근사하는 도형을 그릴 수 있다. 그 후 convexhull 의 corner(harris 기반의 shi-tomasi 방식, 이 방식은 원하는 꼭짓점의 수를 설정할 수 있어 a4 의 네 꼭짓점을 찾기 쉽다)를 구하면 그 점이 a4 의 꼭짓점이다.
Open3d. segment_plane	RANSAC 기반으로 구현된 함수이다. pcd 에서 크기가 가장 큰 평면을 구하고 그 방정식을 반환한다. 이를 토대로 절편을 구해 xy 평면 위에 있도록 transform 시킬 수 있다(개발한 rigid_transform_3d 함수 이용)
Open3d. get_axis_alinged_bounding_box	Point 를 모두 포함하는 부피가 가장 작은 box 를 생성한다. 이때 box 는 axis 에 aligned 되어 있다. 이 box 의 point 를 구해 발의 길이와 폭을 측정할 것이다.
Open3d. hidden_point_removal	물체에 의해 가려지는 부분에 노이즈가 생긴다. 이를 Kinect camera 자체의 보정에 의해 생기는 것으로 추정된다. 이를 해결해기 위해 위의 함수를 적용하면 카메라의 위치에서 볼 때 가려진 부분의 point 즉 노이즈가 제거된다.

표 3. 개발한 코드 내용

파일명: 함수명	기능
Rigid_transform_3D	Kabsch algorithm[vi]을 사용하여 3D 상에서 대응되는 좌표쌍을 입력하면 그에 해당하는 변환행렬이 반환된다.
Match_xy_plane	Linear least squares 를 사용하여 3 차원 point cloud 집합을 선형 평면으로 근사 후 각 x, y 축을 기준으로 회전 이동시킨 point cloud 를 반환한다.
Get_xy_mean_and_eigenvectors	Point cloud 를 입력으로 3 차원 Qhull 의 vertex 집합으로 변환하여 x,y 의 평균, 공분산 행렬을 얻는다. 또한 특잇값 분해를 이용하여 eigenvector 를 얻은뒤 x,y 평균, eigenvector 를 반환한다.

4. 실험 결과 및 분석

2.1 실험 환경 사항

실험 환경은 window10 에서 anaconda 로 설치된 python 3.8.10 version 과 opencv-python 4.5.1, open3d 0.11.2 version 을 사용하였다.

실험에 사용한 데이터는 주변 사람의 발 사진을 직접 촬영하여 사용하였다.

2.2 실험 결과

SOTA 는 앱스토어서 설치할 수 있는 perfitt 앱의 2D color 이미지를 통한 발의 길이와 폭을 측정하여 신발을 추천하는 알고리즘을 사용하였다. 이를 실제 측정값과 우리가 실험한 값과 비교하였다.

SOTA 는 키트로 아래의 그림과 같은 방식으로 측정한다.

표 4. 제안된 알고리즘의 성능분석(단위 : %)

	우리결과	비교 [perfitt]
발 길이	6.32	2.70
발 폭	11.39	4.86
발 높이	12.67	-

보다 자세한 내용은 붙임 참조

추가적으로 촬영의 조건을 달리하여 찍어보았다.

표 5. SOTA 와 다른 각도에서의 결과와 비교

발 뒤꿈치가 보이는 각도						
	길이	발폭	높이	길이 오차	발폭 오차	높이 오차
상민	234	93.5	67.5	1.739%	10.000%	15.625%
영식	248	113	72.5	3.333%	17.708%	14.706%
퍼핏(발 뒤꿈치가 경계에 닿게)						
	길이	발폭	높이	길이 오차	발폭 오차	높이 오차
상민	234.5	106.5	75	1.957%	25.294%	6.250%
영식	256	104	70	6.667%	8.333%	17.647%

2.2 실험 결과 분석

발의 길이는 발의 pcd 가 axis 에 aligned 되었을 때 y 성분의 최대 거리차를 사용하였고, 발의 폭은 x 성분의 거리차의 최대 값을 사용하였다. 발의 높이는 발목 즉 발등과 다리 사이의 경계에서 바닥까지의 거리를 잡았다.

각기 다른 10 명의 발사진을 perfitt 과 같은 위치에서 사진을 2 장 찍은 값의 평균을 구했다. Perfitt 은 양말을 신은 상태로 사진을 찍어 실제 발 사이즈보다 크게 나왔을 것으로 예상된다.

대체적으로 perfitt 이 우리의 실험 결과보다 더 정확하게 나왔다. 이러한 이유를 아래에서 분석할 것이다.

1. 오차 별 발생 이유

기본적으로 발의 실제 측정할 때도 기본적인 오차가 있었다. 측정에 사용된 자가 발과 완전히 평행하지 않아 측정 오차가 발생했다. 특히 발 볼을 측정하는 경우 사람마다 발 뼈의 위치가 달라 기준인 최대거리를 구하기 힘들었고 최대거리로 보인 것이 실제로 최대거리가 아니었을 수도 있다. 특히 높이의 경우 자가 완전히 영점에서부터 측정되지 않고 바닥과 완전히 수직하지 않아서 측정했을 때 오차가 크게 나온 것으로 보인다.

(1) 발 길이 오차

- 카메라 센서의 오차가 있다. 카메라의 왜곡에 의해 발생하는 것으로 카메라 자체의 캘리브레이션을 조절해서 해결하려 했으나 color 의 왜곡이 크고 depth 의 왜곡이 작아서 왜곡 보정 후 1 대 1 매칭을 보장할 수 없기 때문에 이를 통해 해결할 수 없었다. 대신 a4 의 길이의 길이가 일정함을 이용하여 pcd 상에서 a4 의 길이를 측정하여 보정하였다. 그러나 왜곡이 이렇게 1 차원적이지 않아 완전한 해결법이라고 생각이 들지는 않는다.

-또한 측정에 쓰인 Kinect 카메라는 측정 높이에 따라 오차가 크게 변한다. 대체적으로 25cm~ 50cm 정도의 높이에서 찍는 것을 권장[vii]되나 이를 철저히 지키면서 발과 a4 를 온전히 촬영하기 힘들기 때문에 오차가 발생했을 것이다. 또한 카메라 삼각대의 높이가 낮아 카메라를 고정할 수 없어 손으로 들고 찍었고 이 때문에 카메라가 흔들리거나 카메라의 높이가 일정하지 않아 오차가 발생했을 것 같다. 또한 피사체의 재질이나 조명에 따라 오차가 추가적으로 발생된다.

- 본 실험에서는 알고리즘 상 촬영 시 a4 끝에 발 뒤꿈치를 맞추거나, 발 뒤꿈치가 보이도록 촬영하여야 한다. 이를 지키지 않을 경우 뒤꿈치 위치 정보의 부재로 발 길이 측정에 오차가 생기게 된다.

-중간에 color 와 depth 이미지를 찍는 코드가 달라졌다. 처음에는 카메라를 찍을 때 depth 가 0 인 부분은 빈 공간이 되었으나 중간에 코드 수정이 되면서 주변의 값을 갖게 되었다. 때문에 pcd 를 뽑았을 때 point cloud 의 높이에 문제가 생겼다.

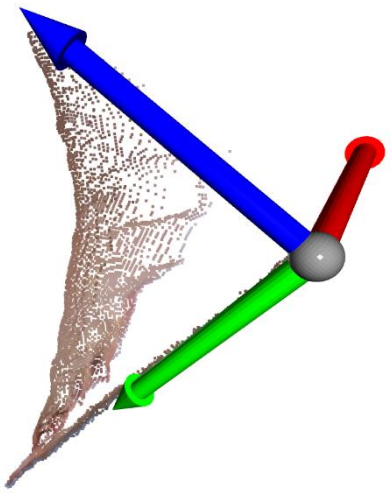


그림 20. xy 축에 정렬한 발 영역 pcd

-발을 xy 축에 정렬하는 과정에서 정확히 정렬되지 않는 경우가 있었다. 때문에 발의 길이와 폭이 실제보다 크게 나오게 되었다.

2. 오차 해결 방안

- 아이폰에서 TOF 카메라로 측정하여 비교해 보려 했으나 관련된 앱을 찾을 수 없었다. ply 파일을 얻을 수는 있었으나 이를 2D 이미지로 역변환하는 코드를 찾을 수 없었다. 추후에 추가적인 연구로 point cloud 로 2D 이미지를 얻거나 point cloud 에서 segment 하는 알고리즘을 구현해 볼 예정이다. 이를 통해 센서에 대한 차이가 있는지, 보정할 수 있는 더 좋은 방식이 있는지를 알아볼 것이다.

- 기존의 코드를 사용해보고 싶었으나 현재 사용환경이 window 이기 때문에, linux 환경의 코드를 수정하기에는 어려움이 있었다. 또한 원본 코드 작성자와 연락이 닿지 않아 어려움이 있었다.

- 발의 모양에 따라 공분산이 달라져 정렬이 제대로 되지 않는 현상이 발생된 것 같았다. 특히

발가락을 완전히 모으지 않은 경우 오차가 더 크게 발생되었다. 이를 해결하기 위해 발의 위치를 강제하는 방법 또한 생각해 볼 수 있겠다.

3. SOTA 와의 비교(장단점 등)

- SOTA 를 통해 발을 측정한 경우 시간이 약 4 초정도 걸렸다. SOTA 의 코드가 완전히 공개되어 있지 않고 앱 형태로 있기 때문에 확신을 할 수 없지만 딥러닝을 사용하여 길이를 측정하는 것으로 알고 있다. 사용된 환경이 같지 않으나 일반적으로 비용이 큰 딥러닝 알고리즘을 사용하지 않고, 전통적 영상처리 알고리즘을 사용한 본 실험 코드가 더 빠르기 때문에 이 점에서 장점이 있을 것이다.

- SOTA 는 정해진 키트 위에서만 찍어야 하고 ROI 를 통해 키트의 위치를 한정시킨다. 우리의 실험은 a4 의 네 꼭짓점만 보이면 되므로 접근성이 용이하다. 또한 발 뒤꿈치의 정보 또한 측정 가능하므로 활용도가 높다.

- 발의 point cloud 를 얻어 측정하므로 3d 정보 예를 들면 높이 정보를 얻을 수 있다. 높이 정보만 아니라 발등의 각도 등 다양한 발의 특성 또한 측정 가능할 것으로 보인다.

- 단점은 SOTA 보다 결과가 좋지 않았다. 이는 우리의 실험이 SOTA 보다 자유로운 환경이어서 그랬던 것으로 보인다. 또한 SOTA 를 측정할 때 핸드폰 삼각대를 사용하여 흔들림이 없어 오차가 크지 않게 나왔던 것 같다. 또한 사용한 알고리즘에서의 차이도 있을 것이다. 위의 단점은 단순 추측이고, 원인을 알기 힘들다. 따라서 추가적인 실험을 더욱 많이 하여 개선해야 한다.

4. 앞으로의 개선방안

- 2D segment 를 할 때 depth 정보를 사용하지 않았고 기하적인 정보 대신 color 정보에 의존하였다. 이는 환경에 영향을 많이 받으므로 지양해야 한다. Segment 자체의 시간이 그리 길지는 않지만(약 0.1 초,붙임 4 참조) 만약 2D 에서 하게 되면 다른 접근 방법이 더 좋을 것 같다. 2D segment 에서의 핵심은 a4 를 찾는 것인데 이를 딥러닝을 통해 해결하면 더 정확한 결과가 나올 것으로 보인다.
다만 붙임.4 에서 분석 했다시피 rgbd image 를 pcd 로 변경하는데 대부분의 시간이 소요되었다. 대부분의 카메라 예를 들면 Kinect, Iphone TOF 카메라는 ply 파일을 계산한다. 이를 이용해 ply 파일에서 segment-2D 에서 하는 것보다 시간이 더 오래 걸리겠지만-하는 것이 더욱 효율적일 것 같다. 이 경우 depth 정보를 사용할 수 있고 과도하게 color 의존적이지 않게 되어 더욱 다양한 환경에서 정확한 정보를 얻을 수 있을 것으로 보인다.

- 아이폰이나 다른 TOF 카메라 또는 depth 정보를 얻을 수 있는 여러 장비들을 통해 어떤 환경에서 카메라 오차가 적게 나오는지 확인하여 최적의 측정 환경을 찾아본다.

- 사진의 촬영방식에 따라 가려지는 발들의 정보가 생길 수밖에 없다. 따라서 여러 각도에서 찍은 사진들을 접합하는 방식을 시도하면 더 정확하고 많은 발의 정보를 가진 3d 모델을 얻을 수 있을 것이다.

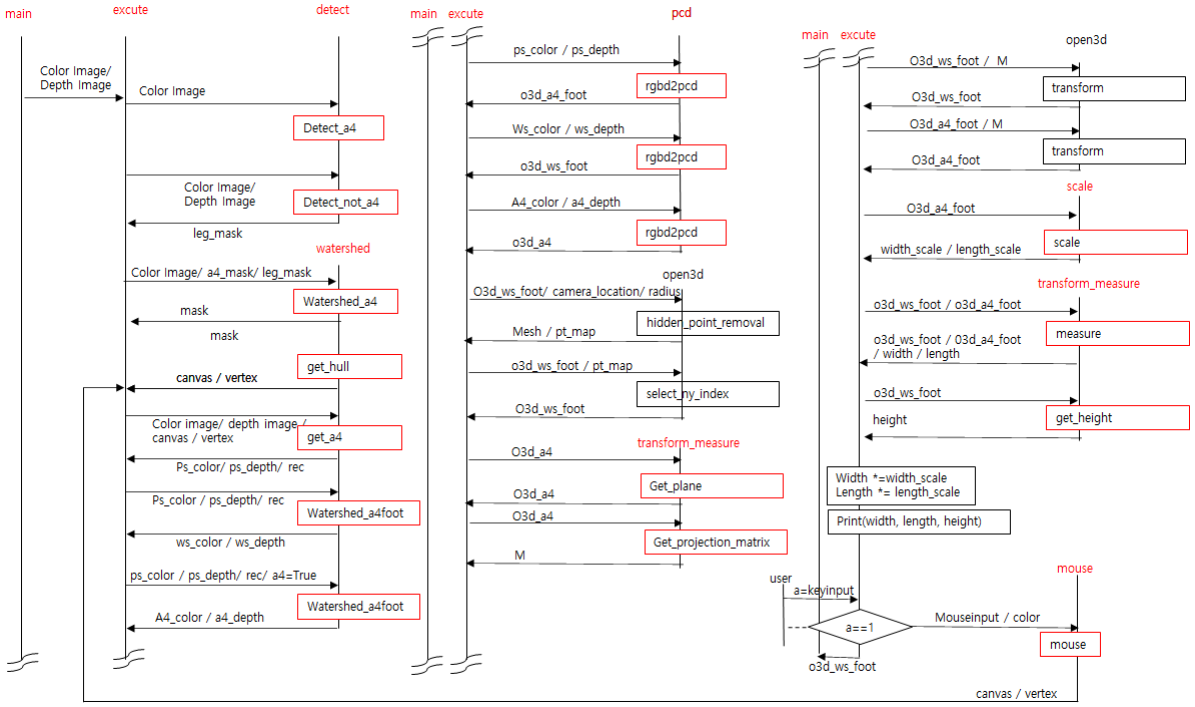
5. 조원 역할

표 6. 조원 역할

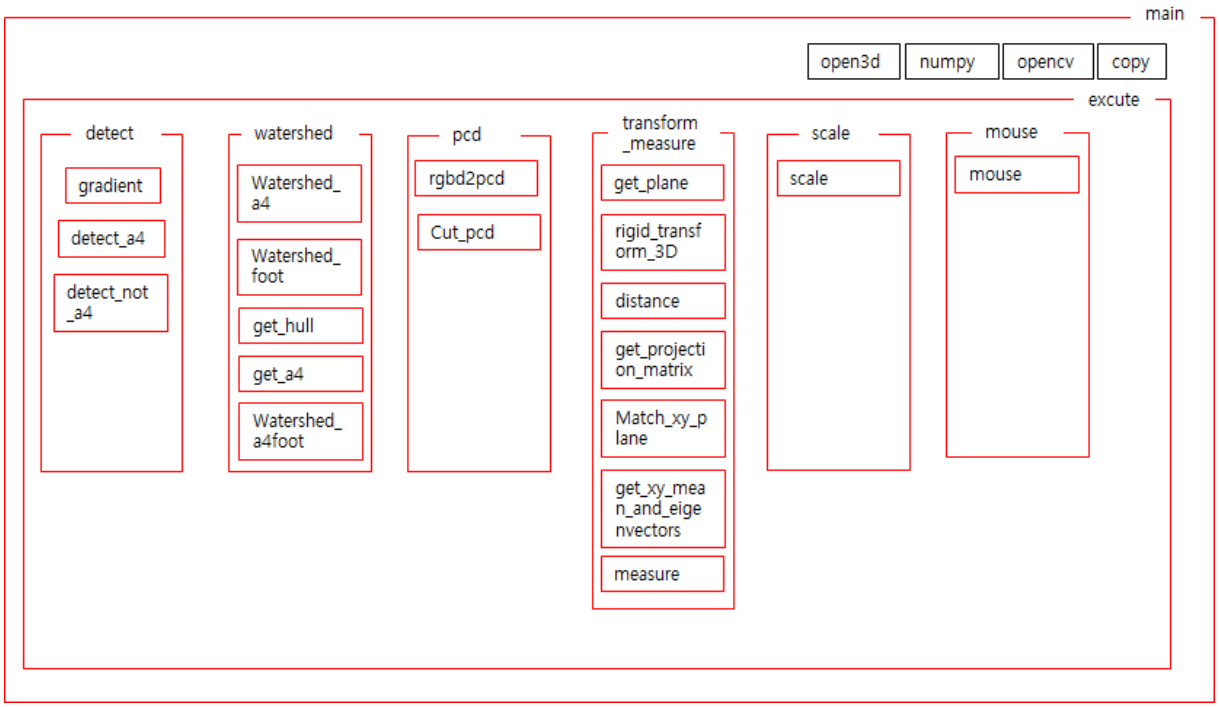
조원	알고리즘/ 구현
윤영식	- 2d segment 알고리즘 개발 - Plane 의 절편을 이용한 transform - A4 의 크기를 통한 scale 보정 알고리즘 개발
오상민	- 공분산을 통한 eigenvector, eigenvalue 계산, 이를 활용한 transform 알고리즘 개발 - Bounding box 를 통한 발의 길이와 폭 측정 알고리즘 개발

붙임

1. 블록 다이어그램



2. python 파일 계층 구조



3. 알고리즘 성능 분석

실제			perfitt				실험값					
길이	발폭	높이	길이	발폭	길이 오차	발폭 오차	길이	발폭	높이	길이 오차	발폭 오차	높이 오차
230	85	80	224.5	91.5	2.391%	7.647%	234.5	106.5	75	1.957%	25.294%	6.250%
250	95	80	256.5	91	2.600%	4.211%	254.5	101	72.5	1.800%	6.316%	9.375%
235	95	85	241	98.5	2.553%	3.684%	238.5	102.5	70	1.489%	7.895%	17.647%
268	101	80	265	104.5	1.119%	3.465%	226.5	128.5	72.5	15.485%	27.228%	9.375%
265	100	75	269.5	101	1.698%	1.000%	276	124	75	4.151%	24.000%	0.000%
240	96	85	244	100.5	1.667%	4.688%	256	104	70	6.667%	8.333%	17.647%
242	100	95	246.5	101.5	1.860%	1.500%	261.5	109.5	85	8.058%	9.500%	10.526%
230	95	95	248	91.5	7.826%	3.684%	268.5	104	72.5	16.739%	9.474%	23.684%
266	99	77	266	99	0.000%	0.000%	266.5	99.5	77.5	0.188%	0.505%	0.649%
220	87	90	226	96.5	2.727%	10.920%	242	84.5	75	10.000%	2.874%	16.667%
245	103	100	258	116	5.306%	12.621%	252.5	107	72.5	3.061%	3.883%	27.500%

4.알고리즘의 시간 분석(cProfile, 분석을 안 해도 되거나 무시해도 될만큼 작은 값들은 삭제하였다)

1642 function calls (1617 primitive calls) in 1.060 seconds

Ordered by: standard name

ncalls	tottime	percall	**cumtime**	percall	filename:lineno(function)
4	0.003	0.001	0.003	0.001	copy.py:128(deepcopy)
1	0.003	0.003	0.019	0.019	detect.py:11(detect_a4)
1	0.001	0.001	0.002	0.002	detect.py:50(detect_not_a4)
2	0.005	0.002	0.014	0.007	detect.py:6(gradient)
2	0.000	0.000	0.000	0.000	linalg.py:102(get_linalg_error_extobj)
1	0.000	0.000	0.000	0.000	pcd.py:31(cut_pcd)
3	0.758	0.253	0.769	0.256	pcd.py:6(rgbd2pcd)
1	0.037	0.037	0.039	0.039	scale.py:8(scale)
1	0.001	0.001	0.001	0.001	transform_measure.py:133(get_xy_mean_and_eigenvectors)
1	0.001	0.001	0.004	0.004	transform_measure.py:141(measure)
1	0.002	0.002	0.004	0.004	transform_measure.py:16(rigid_transform_3D)
1	0.006	0.006	0.010	0.010	transform_measure.py:209(get_height)
5	0.000	0.000	0.000	0.000	transform_measure.py:39(distance)
1	0.058	0.058	0.062	0.062	transform_measure.py:46(get_projection_matrix)
1	0.068	0.068	0.068	0.068	transform_measure.py:8(get_plane)
7	0.000	0.000	0.000	0.000	twodim_base.py:156(eye)
8	0.000	0.000	0.000	0.000	watershed.py:102(<lambda>)
1	0.001	0.001	0.009	0.009	watershed.py:126(get_a4)
2	0.002	0.001	0.016	0.008	watershed.py:161(watershed_a4foot)
1	0.001	0.001	0.017	0.017	watershed.py:7(watershed_a4)
2	0.001	0.000	0.007	0.004	watershed.py:74(get_hull)
56	0.000	0.000	0.000	0.000	{arcLength}
9	0.002	0.000	0.002	0.000	{bitwise_and}
8	0.001	0.000	0.001	0.000	{bitwise_or}
157/132	0.004	0.000	0.015	0.000	{built-in method numpy.core._multiarray_umath.implement_array_function}
3	0.001	0.000	0.001	0.000	{calcBackProject}
4	0.000	0.000	0.000	0.000	{calcHist}
2	0.000	0.000	0.000	0.000	{circle}
6	0.007	0.001	0.007	0.001	{connectedComponentsWithStats}
4	0.000	0.000	0.000	0.000	{convexHull}
11	0.002	0.000	0.002	0.000	{cvtColor}
4	0.000	0.000	0.000	0.000	{drawContours}
2	0.000	0.000	0.000	0.000	{findContours}
10	0.000	0.000	0.000	0.000	{getStructuringElement}
2	0.005	0.003	0.005	0.003	{goodFeaturesToTrack}
3	0.001	0.000	0.001	0.000	{imread}
23	0.005	0.000	0.005	0.000	{method 'astype' of 'numpy.ndarray' objects}

참고문헌

[i] 통계청 -온라인 쇼핑몰 취급상품범위별 상품군별 거래액: https://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_1KE10041&conn_path=I3 (21. 06. 12 접속)

[ii] Perfit 기업 소개: https://www.perfitt.io/ (21. 06. 12 접속)

[iii] Cv2.watershed reference : https://docs.opencv.org/master/d3/d47/group__imgproc__segmentation.html#ga3267243e4d3f95165d55a618c65ac6e1 (21. 06. 12 접속)

[iv] Cv2.convexhull reference: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga014b28e56cb8854c0de4a211cb2be656 (21.06.12 접속)

[v] Open3d document : http://www.open3d.org/docs/release/tutorial/geometry/pointcloud.html (21. 06. 12 접속)

[vi] Kabashi argorithm : http://nghiaho.com/?page_id=671(21. 06. 12 접속)

[vii] Kinect 작동환경 : https://docs.microsoft.com/ko-kr/azure/kinect-dk/hardware-specification (21. 06. 12 접속)