4. 본 연구 방법(실험 부분)

- 전방 객체 인식

Object Detection의 알고리즘으로 실시간으로 구현이 가능한 YOLOv5 알고리즘을 사용하였다. YOLOv5 알고리즘은 기존의 YOLO 버전이 낮은 알고리즘 보다 FPS와 AP의 성능을 높혔다.

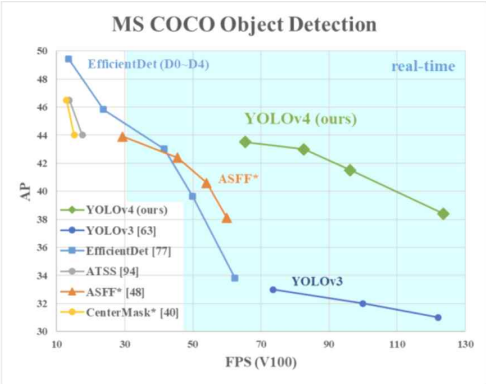

그림 1 YOLOv4

YOLOv4 알고리즘은 위의 그래프와 같이 AP 약 44에 FPS는 70~127 정도의 성능을 보인다. 반면에 YOLOv5 알고리즘은 YOLOv5-x 기준 AP는 66.6에 FPS는 153의 성능을 보인다.

Model	AP^{val}	AP^{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}	params	FLOPs
YOLOv5-s (ckpt)	35.5	35.5	55.0	**2.1ms**	**476**	7.1M	12.6B
YOLOv5-m (ckpt)	42.7	42.7	62.4	3.2ms	312	22.0M	39.0B
YOLOv5-l (ckpt)	45.7	45.9	65.1	4.1ms	243	50.3M	89.0B
YOLOv5-x (ckpt)	**47.2**	**47.3**	**66.6**	6.5ms	153	95.9M	170.3B
YOLOv3-SPP (ckpt)	45.6	45.5	65.2	4.8ms	208	63.0M	118.0B

그림 2 YOLOv5

이러한 성능을 끌어올린 원인은 Backbone과 Architecture의 차이에 있다.

Backbone 부분은 이미지로부터 Feature Map을 추출하는 부분으로 YOLOv4와 유사하다. 하지만 CSPNet을 활용하면서 CNN의 학습 능력을 향상시켰다. 또한, Architecture 부분은 BottleneckCSP를 추가 활용하며 성능을 향상시켰다.

- Optical Flow Estimation

Optical Flow는 광학 흐름 또는 광학 흐름이라고 불리며 관찰자와 장면 사이의 상대적인

움직임으로 인해 시각적 장면에서 물체, 표면 및 가장자리의 명백한 움직임의 패턴이다.

최근 딥러닝을 활용한 Optical Flow의 개발(ex. FlowNet)이 주를 이루고 있지만 YOLOv5 알고리즘을 적용했을 때 연산량이 많아지면서 실시간성의 구현이 어렵다. 이의 한계점을 극복하기 위해 딥러닝을 활용하지 않은 Optical Flow 알고리즘을 사용한다.

첫 번째로 Block Matching 알고리즘과 Horn-schunck 알고리즘이 있다. 두 알고리즘은 정확도가 높지만 계산량이 매우 많아 실시간의 구현이 어렵고 OpenCV 3 버전 이상부턴 사용이 불가하여 YOLOv5 알고리즘과 연동이 불가하다.

OpenCV 4 버전 이상에도 동작하는 알고리즘 중에는 크게 2가지 알고리즘이 있다.

첫 번째로 Lucas-Kanade Pyramid 알고리즘이다. 이 알고리즘은 기존의 Lucas-Kanade 알고리즘에선 큰 움직임의 추출이 불가하단 점을 보완한 알고리즘이다.

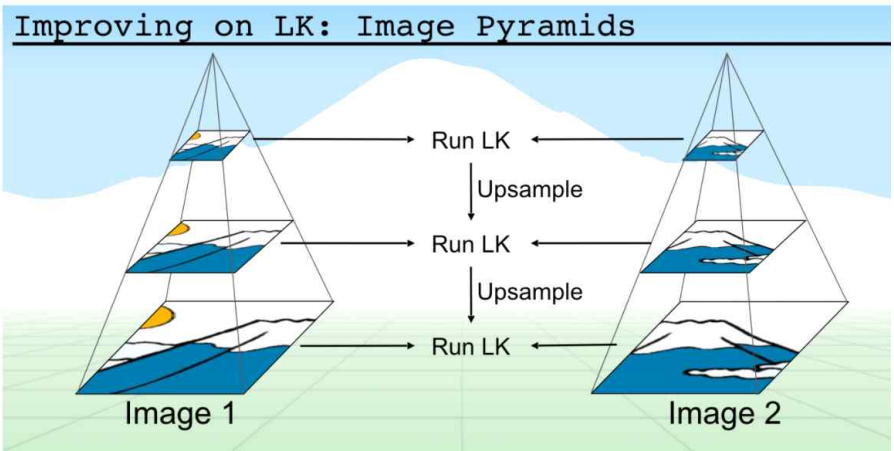

그림 3 Lucas-Kanade Pyramid

위 그림과 같이 이미지 사이즈를 조정하고 마치 피라미드처럼 쌓아서 이미지의 크기를 작게 하여 계산된 변위부터 차례로 Lucas-Kanade 알고리즘을 적용하여 변위를 계산하는 방식이다. 하지만 이 알고리즘은 특징점을 찾은 후 계산하기 때문에 특정 영역만을 보고 실제 움직임을 잘못 판별할 수 있는 barberpole illusion이 발생할 수 있다.

두 번째로 Gunner가 개발한 Farneback Optical Flow 방식이다. 몇몇 feature에 대해서만 flow를 계산하는 sparse한 방식과는 달리 Farneback Optical Flow 방식은 dense하게 모든 픽셀에 대해서 flow를 계산한다. 이 알고리즘은 테일러 방정식을 2차 다항식으로 이미지 프레임의 창을 근사화한다. 그 이후 움직임하에서 다항식이 어떻게 변환되는지 관찰하며 다항식 확장 계수에서 변위 필드를 추정하는 방법을 수행한다. 모든 픽셀에 대해서 위와 같은 연산이 일어나 Lucas-Kanade보다 속도가 느린 단점이 있다. 그러나 모든 픽셀에 대해서 계산하기 때문에 정확도가 더 뛰어나다.

다음 표는 현재 Opencv 4 이상에서 적용할 수 있는 알고리즘들에 대해 소요된 시간과 그에 따른 결과 사진이다. 영상은 PETS2009 Benchmark 비디오를 이용하였으며, Opencv에서 예제로 제공되었다. Simpleflow 알고리즘은 세밀한 Optical Flow를 나타낼 수 있지만 많은 시간이 소요되며, DenseRLOF는 Optical Flow의 결과값이 좋지 않았으며, Sparse to dense는 화질이 좋은 영상(4K)에서 적용할 수 없는 단점이 있다.

표 1 실험 실행 환경

	정보
CPU	i9-10850K, 3.60GHz 10 core 20 threads
RAM	16GB
영상정보	width = 480, height = 360 playtime = 79s frame rate = 10

표 2 각 알고리즘의 소요되는 시간

	Time(grid)[s]	Time(HSV)[s]
Simpleflow	211.87	189.40
Gunner - Farnerback algorithm	42.04	37.76
DenseRLOF	74.57	40.27
Sparse to dense	40.61	39.56
Lucas-Kanade algorithm	15.89	

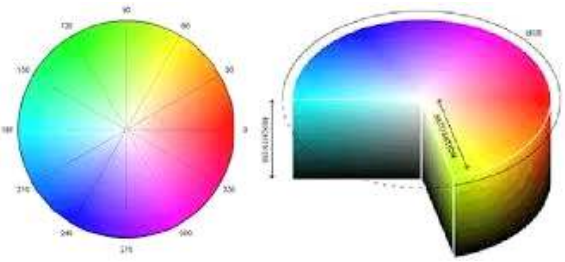

그림 4 HSV

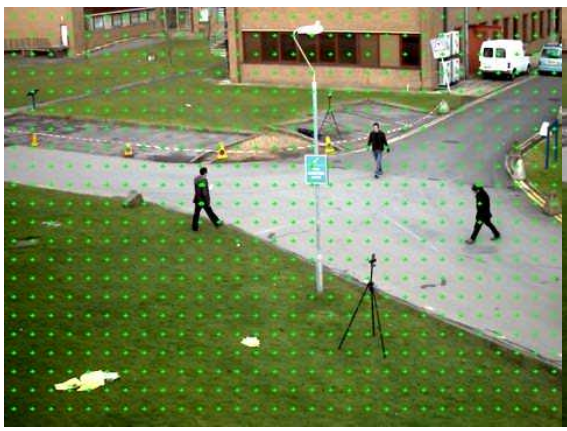

그림 5 Simpleflow, grid

그림 6 Simpleflow, HSV

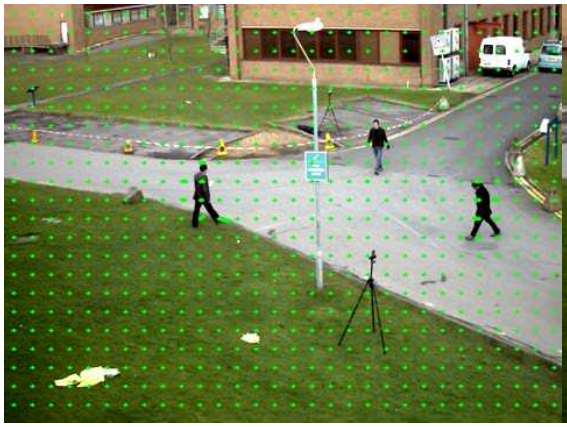

그림 7 Gunner Farnerback, grid

그림 8 Gunner Farnerback, HSV

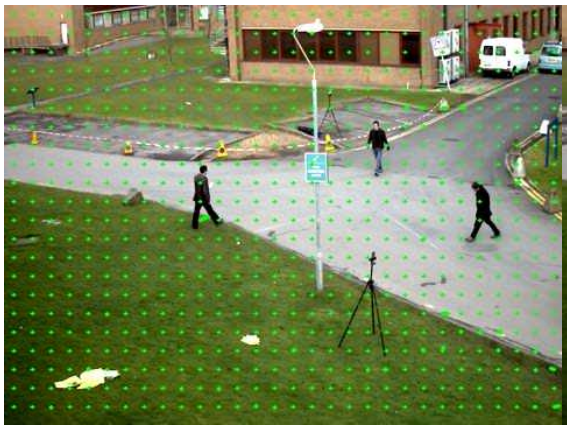

그림 9 DenseRLOF, grid

그림 10 DenseRLOF, HSV

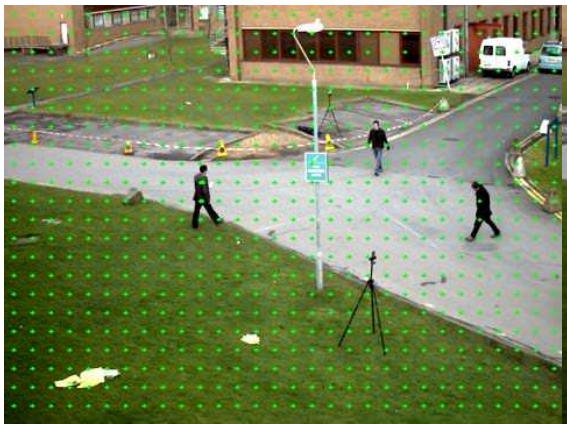

그림 11 Sparse to dense, grid

그림 12 Sparse to dense, HSV

그림 13 Lucas-Kanade algorithm

- 제안 방법

1) 전체적인 흐름

실시간성과 정확성을 높이기 위해 YOLOv5 알고리즘과 Farneback 알고리즘을 사용한다. Farneback 알고리즘이 연산량이 많아 속도가 느리다는 단점을 극복하기 위해 Image Resizing 방식을 이용한다.

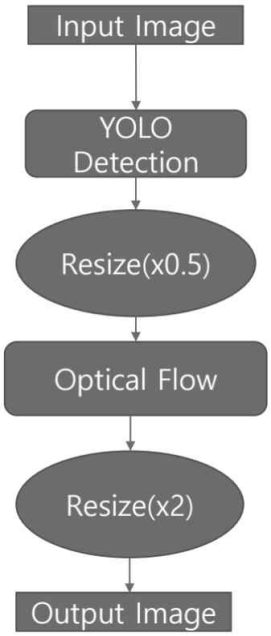

그림 14 Flow Chart

YOLOv5 알고리즘을 통해 Object Detection을 한 후 Detection한 이미지를 반으로 Resize 한다. 그 이후 Optical Flow 계산을 통해 픽셀에 대한 연산값을 줄인다. 계산이 끝난 후 다시 Resizing 하여 결과 이미지를 출력하게 되는 방식이다.

2) Optical flow를 통한 대표 벡터 추출

Farneback 알고리즘은 모든 픽셀값을 계산하기 때문에 결과를 표현할 때도 모든 픽셀의 각도와 크기를 계산하여 색을 이용해 보여준다. 하지만 이는 직관적으로 객체가 어느 방향과 어느 속도로 이동하는지를 모르기 때문에 대표 벡터를 추출한다. 이 대표 벡터를 추출하는 방법으로 제안하는 것은 다음과 같다.

$$X = magnitude \times \cos(\theta)$$
$$Y = magnitude \times \sin(\theta)$$

픽셀 별 각도와 크기를 X,Y 벡터로 변환해준 후 리스트로 저장한다. 그리고 bounding box 영역 내에 있는 모든 벡터 리스트의 평균을 구한다. 이를 각 프레임 하나 당의 대표 벡터라 하고 이 대표 벡터를 5 프레임을 쌓아 벡터의 합을 계산한다.

$$magnitude = ||X, Y||$$
$$\theta = \arctan(Y/X)$$

계산한 벡터의 합을 통해 객체의 움직임의 크기와 각도를 추출한다.

3) 충돌 여부 탐지

충돌을 할 때 고려할 조건은 객체와의 거리와 객체가 오는 각도에 따라 달라진다. 먼저 각도에 따른 충돌 위험도를 계산하기 위해 탐지된 객체가 어느 위치에 있는지를 탐지한다. 이를 탐지하는 이유는 객체의 위치마다 충돌할 수 있는 각도가 다르기 때문이다. 위치 탐지 방법은 다음과 같다.

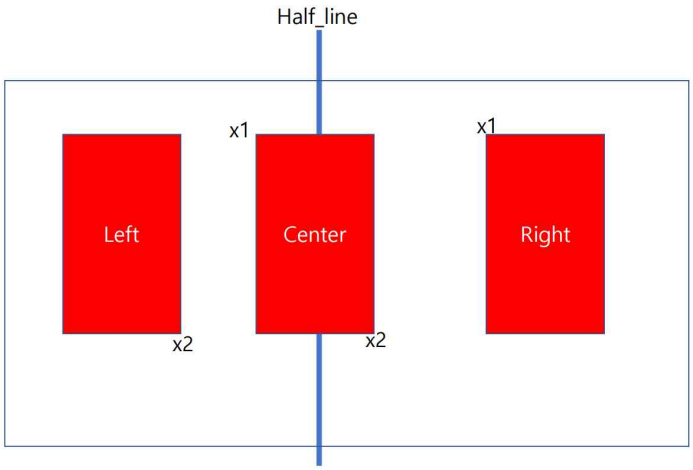

그림 15 객체 위치 탐지 방법

이미지를 반으로 나눈 후 bounding box의 끝 점 좌표에 따라 다르게 설정하였다. bounding box의 우하단 모서리의 좌표가 하프라인의 x 좌표보다 작으면 왼쪽, 좌상단 좌표가 하프라인의 x 좌표보다 크면 오른쪽, 좌상단의 좌표는 하프라인의 x좌표보다 작지만 우하단 좌표는 크

다면 가운데에 있다고 정의한다. 그리고 탐지된 위치에 따라 충돌이 예상되는 각도의 범위를 다르게 할당한다.

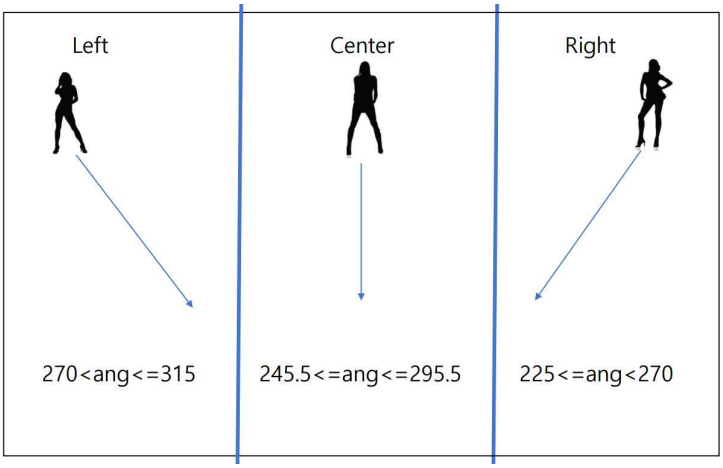

그림 16 객체 위치에 따른 충돌 각도

각도의 단위는 degree로 통일한 후 충돌 각도 설정을 한다. 객체가 왼쪽에 있을 때는 270도 초과 315도 이하, 객체가 오른쪽에 있을 때는 225도 이상 270도 미만, 가운데에 있을 때는 245.5도 이상 295.5도 이하인 각도를 충돌 각도라고 설정한다. 객체의 움직임이 충돌 각도를 나타낼 때는 위험 알람을 주는 조건 중 하나가 된다.

각도만으로는 충돌 여부를 알 수 없기 때문에 객체와의 거리를 기하학적인 방법을 통해 추정한다. 2D 이미지의 한계점을 기하학적인 계산 방법으로 객체와의 거리를 추정하였다.

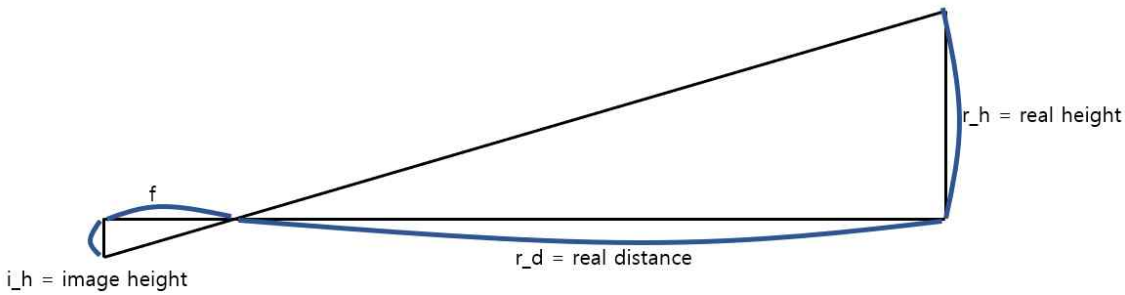

그림 17 카메라와 물체 간 거리 계산 방법

카메라의 초점 거리를 f, 실제 물체의 높이를 r_h(real distance), 이미지 상에서의 물체의 높이를 i_h 라고 할 때, 실제 거리 r_d를 구하기 위해서는 다음과 같은 닮음을 이용한 식을 이용한다.

$$i_h : f = r_h : r_d$$

$$r_d = \frac{f r_h}{i_h}$$

i_h를 픽셀 단위에서 mm단위로 변환하기 위해서 이미지의 전체 세로 길이(pixel)로부터 물체의 세로 길이를 구한 다음, 이미지의 세로 길이(mm)를 곱하여 계산한다.

$$i_h = \frac{O_{\mathrm{h}}}{I_{\mathrm{h}}} s$$

$$r_d = \frac{f r_h I_{\mathrm{h}}}{O_{\mathrm{h}} \times s}$$

I_{h}는 이미지의 전체 세로 길이(pixel), O_{h}는 객체의 세로 길이(pixel), s는 이미지의 세로 길이(mm)를 뜻한다.

위의 식에서 f = 18(mm), 실제 높이 = 1750(mm), 이미지 전체 세로 픽셀 개수 = , 이미지의 세로 길이 = 25(mm)로 측정되었으며 이 식을 이용하여 카메라와 물체 간 거리를 측정할 수 있다.
만약 사람이 가까워져 전체 모습이 카메라에 맺히지 않는 거리가 된다면 그에 따른 보정을 하게 되는데, bounding box의 x 좌표의 차이(O_w)가 계속 커지면 사람이 계속 가까워지고 있음을 알 수 있으므로 위 식에서 카메라에 담지 못하는 세로 픽셀을 보정한다.
점점 가까워질수록 O_h에 대해 ((1 + 0.05α)의 가중치 + 0.02α O_w)를 두었으며, σ는 x가 커지는 경우 전체 모습이 맺히지 않은 경우 더 큰 값을 가지도록 설계하였다.

$$r_{d'} = \frac{f r_h I_h}{(0.02\alpha O_w + (1 + 0.05\alpha) O_h) s}$$

또한 위 식과 아래 식을 이용하여 더 좋은 거리 예상 모델을 위해 사람의 전체 모습이 카메라에 맺히지 않는 경우 아래의 식을 이용한다.

$$(1 - \frac{1}{\alpha}) r_{d'} + (\frac{1}{\alpha}) r_d$$

위와 같은 과정을 통해 구한 객체와의 거리와 객체의 방향 각도를 이용하여 각도의 범위가 충돌 위험 각도이고 객체와의 거리가 1m 이하일 때 경고 메시지를 띄우도록 설정을 하였다.

충돌 예상 시간(TTC, Time To Collision)은 움직이는 물체에 대한 상대속도를 이용하여 계산할 수 있다. 위의 과정에서 움직이는 물체에 대한 거리를 구하였으므로 물체에 대한 상대 평균 속도 v는 다음 식으로 알 수 있다.

$$\frac{\Delta r_d}{\Delta t} = v$$

Δr_d는 두 프레임에서 움직이는 물체에 대한 거리의 차, Δt는 두 프레임 간의 시간 차이다.

4) 실험 결과

다음은 위 방식을 이용하여 실험한 결과이다. 탐지된 물체는 bounding box로 나타냈으며, bounding box 위의 거리는 카메라와 물체 간의 거리를 나타낸 것이다. 빨간 화살표는 탐지된 물체의 Optical Flow의 평균이다. 충돌하지 않는 경우 “Safe!“, 충돌하는 경우 ”Warning!“, 충돌하는 경우 예상되는 충돌시간(TTC, Time To Collision)을 표시하였으며, 충돌 예상 지점(PCP, Predicted Collision Point)을 파란 점으로 표시하였다.

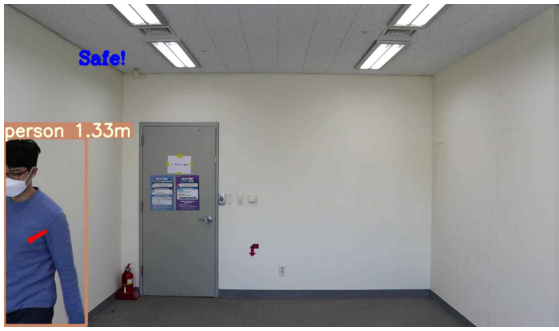

그림 18 왼쪽에서 충돌하지 않는 객체

그림 19 왼쪽에서 다가오는 충돌하려는 객체

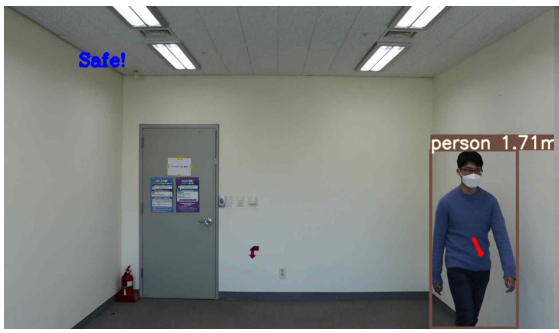

그림 20 오른쪽에서 충돌하지 않는 객체

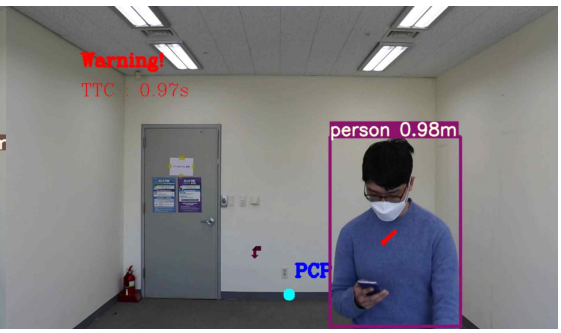

그림 21 오른쪽에서 다가오는 충돌하려는 객체

그림 22 중앙에서 충돌하기에 멀리 있는 객체

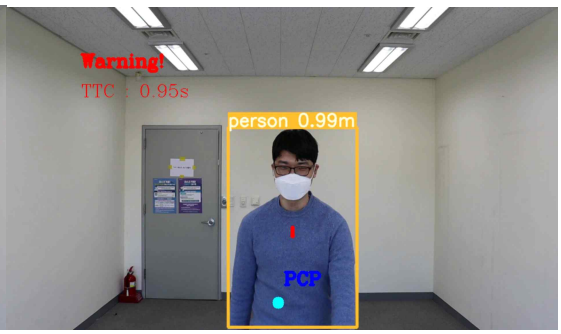

그림 23 중앙에서 충돌하려는 객체