종합설계 1조

고열자 탐지 무인 시스템

12121122 변재철
12141533 박대현
12171588 김민석

1. 개요

1.1 소개

- 고열자 탐지 무인 시스템은, 고열을 탐지해야하는 환경에서, 고열자를 탐지하고 그것을 클라이언트에게 수신하여 무인 상태에서도 사용할 수 있도록 하는 시스템이다.
- 장치에서 고열자를 탐지할 경우, 클라이언트는 UI를 통해 해당 고열자의 사진을 받아볼 수 있고, 메세지를 보내 이동을 중지시키고, 행동을 제한할 수 있다
- 기존 고열자 시스템이 사람이 관리해야하고, 유인 시스템이므로 이에 따른 문제가 언론에서 지속적으로 지적되고 있는 바, 무인으로 관리될 경우 책임 소재가 명확하게 할 수 있고, 관리하기에도 훨씬 용이하다.

1.2 개발 목적

- 국내에 열화상 카메라가 도입되어 운용 중에 있다.
- 하지만 유인 시스템으로서, 인력 및 비용 손실이 상당할 뿐 아니라, 유인 시스템인만큼, 사람의 관리 한계 상 정상적인 기능을 하지 못할 때가 있다.
- 시중에 보급되는 열화상 카메라의 가격이 600만원대에 달하는 만큼, 경제적인 문제가 발생하여 시중에 수요 대비 공급이 부족한 상황이다.
- 이에 경제적이고, 유인 시스템 대신 무인 시스템 구축하는 방법이 필요함을 인식했다.
- 저가의 열상 카메라와 실상 카메라를 구현하고, 사람이 배치되지 않더라도 이를 원격으로 관리할 수 있을 것이다.
- 건물 내의 좁은 복도, 문 등에 설치될 것이기 때문에, 매우 빠른 시간 내에 객체가 사라질 수 있으므로, 실시간 처리가 보장되어야 한다.
- 실시간 처리가 되어야 하므로, 30fps 이상의 처리 속도를 보장하여야 한다.
- 타 제품은 특정 거리의 값으로 한정하여 온도를 측정하다는 약점이 있고, 경제성이 문제가 있다. 또한 무인 시스템을 적용하면, 재화 낭비를 줄이고, 관리를 용이하게 한다.

1.3 팀 소개

	변재철	박대현	김민석
담당 업무	■ UI 개발 및 보조 개발	■ 실상 카메라 데이터 활용 ■ 객체 인식 모델 학습 ■ 보정 및 센서 퓨전 알고리즘 개발	■ 열상 카메라 데이터 활용 ■ 스피커 연동

표 1. 팀 소개

1.4 개발 장비

1.4.1 실상 카메라

1.4.1.1 설명

그림 1. SONY XPERIA XP

SPECIFICATION	
Lens	24mm f/2.0 광각 렌즈
Sensor	Sony Exmor RS IMX300 1/2.3" CMOS sensor

표 2. SONY XPERIA XP Specification

- 실상 이미지를 얻기 위해서는 카메라가 필요하다. 이에 웹캠이 필요하며, 이를 위해 스마트폰 중 하나인 SONY사의 XPERIA X Performance를 사용한다. 해당 제품은 1900만 해상도를 가지며, IMX300 센서를 가지고 있어, 웹캠으로서 사용하는데 충분한 성능을 가지고 있다.
- 해당 스마트폰을 사용하여 DroidCam 어플리케이션을 활용한다. 해당 어플을 사용할 경우, 스마트폰을 웹캠으로 대체하여 사용할 수 있으며, 이는 하드웨어들을 합치어 사용하기에 좀 더 편리하다.
- 열화상 카메라의 해상도의 비율이 4:3이므로, 비율을 맞추기 위하여, 실상 카메라 또한 4:3의 비율로 작동하도록 만든다. 해당 카메라는 640x480의 해상도로 작동하게 된다.
- 실제 사용 시에는 공인 IP를 할당 받으므로, 포트 포워딩을 사용하여 접속한다.

1.4.1.2 프로세스

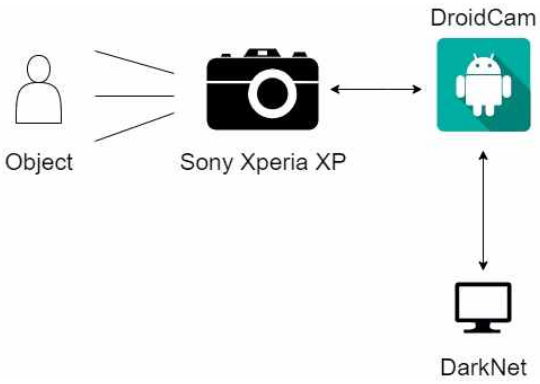


그림 2. 실상 카메라 프로세스

① 웹캠(SONY XPERIA XP)에서 DroidCam Application을 실행하여 해당 기기를 웹캠으로 만든다.
② 객체 인식 용도의 기계학습 툴 중 하나인 Darknet이 DroidCam에 연결한다.
③ 이 때, 웹캠에 연결하기 위해서는 사전에 포트포워딩(Port Forwarding)이 처리되어 있어야한다.
④ DarkNet이 웹캠을 통해 실상 이미지를 얻는다.

1.4.2 열상 카메라

그림 3. 열상 카메라 MLX90640

Specification	
Target temperature	-40°C to 300°C
Accuracy	±1°C
Resolution	32 x 64 pixels
Field of view	55° x 35°
FPS	Up to 64 FPS

표 2. MLX90640 Specification

1.4.2.1 설명

- MLX90640 열화상 카메라를 라즈베리파이에 연동하여 사용하였다.
- MLX90640은 타 제품 대비 저렴한 가격, 높은 hz, 낮은 error rate를 가지고 있지만, 비교적 낮은 해상도를 가지고 있다. 고열자를 판단하기 위해서는 온도의 정확도 면에서 높은 성능을 보여야 하며, 실시간 처리를 위해서 FPS가 30이상인 카메라를 선정하였다.
- 80x60 이상의 해상도를 가진 제품들의 가격을 감안해보면, 저가의 목표를 구현하기에 현실적인 가격이다.
- MLX90640을 라즈베리파이에 연결한 후, C/C++ 코드로 열상 데이터를 추출하였다.
- 카메라로 열화상 이미지를 인식한 후 온도데이터 추출하였다.
- 해상도 부분에서 성능이 높으면 좋겠지만, 그렇지 않을 경우 비교적 좁은 공간인 복도나 문의 출입구 등에 설치하여 인식 거리를 줄인다.
- 물체에 따라 복사에너지의 방사율이 다르다. 따라서 신체 피부의 반사율인 0.98을 열상 카메라에 적용했다.

1.4.2.2 프로세스

① 초기 카메라 설정 및 연결을 set_Thermal_camera()을 통해 수행 ② 프레임의 열상데이터를 get_frame_data() 함수를 통해 습득

1.4.3 거리 센서

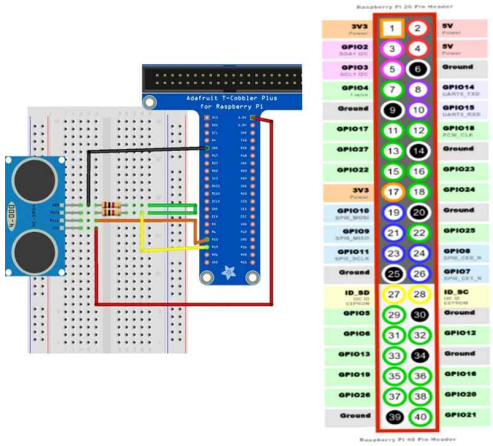


그림 4. 라즈베리파이와 거리 센서 간의 핀 배열

1.4.3.1 설명

- 초음파를 생성하여 쏜 다음, 돌아올 때 드는 시간으로 거리를 측정 - 위의 그림과 같이 라즈베리파이와 연결 - 열상 카메라의 거리에 따라 복사 에너지 손실로 인해 온도를 제대로 측정하지 못하는 문제가 발생. 복사 에너지를 계산하기 위해서는 적외선 캘리브레이터가 필요하지만, 일반적이 가격이 아니어, 거리 센서를 통한 보정을 사용하기로 함.

1.4.4 영상 처리 서버

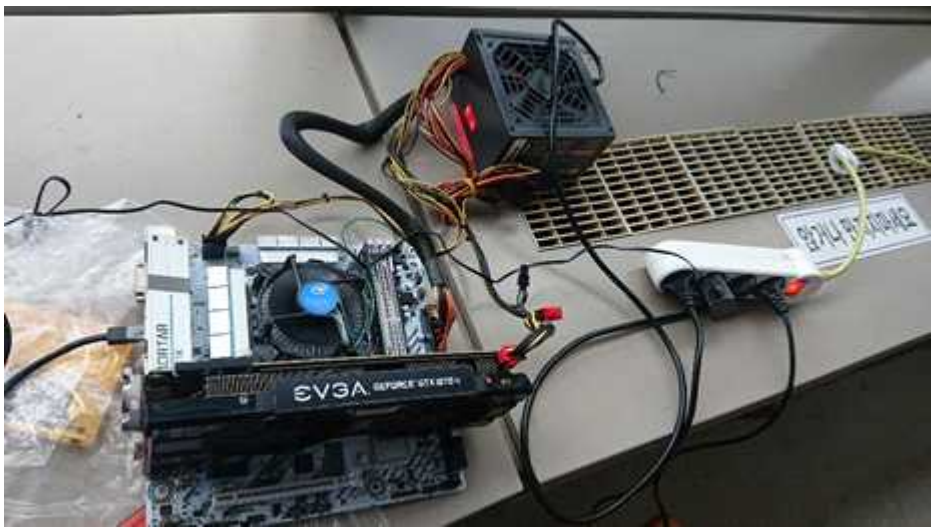


그림 5. 영상 처리 서버 실물

CPU	쿼드코어 CPU (Intel i3-9100F)
RAM	DDR4 16GB
VGA	6.5 TFLOPS (FP32)의 성능을 가진 GPU (Geforce GTX 1070TI 8GB, but non-tensor core)

표 3. 영상 처리 서버 Specification

1.4.4.1 설명

- Ubuntu 18.04 LTS 운영체제를 사용한다.
- 영상 처리 서버는, 객체 인식을 위해 기계 학습을 진행하기 위하여, cuda를 사용해야한다.
- CUDA를 사용하기 위해 NVIDIA사의 그래픽 카드가 필수적이다.
- 영상 처리를 위해서는, 높은 퍼포먼스가 필요하며, 이는 라즈베리파이 같은 소형 컴퓨터에서 처리하기에는 리소스가 너무 많이 들어가게 된다. 따라서 이를 별도로 처리해줄 수 있는 서버가 필요하다.
- 실시간 처리를 위해 최소 30fps 이상의 처리 속도를 가진다.

2. 기능

2.1 전체 프로세스

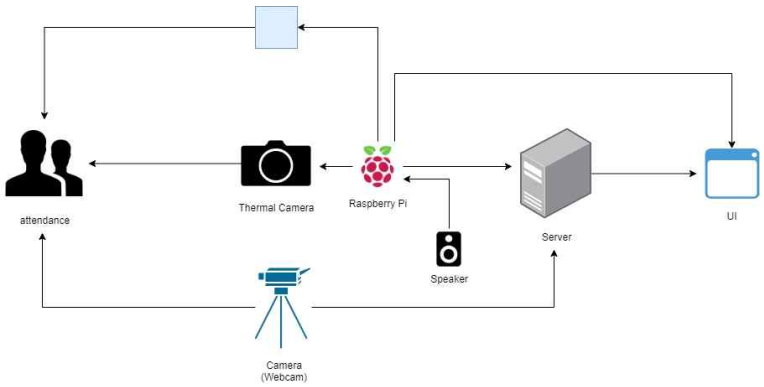


그림 6. 전체 프로세스

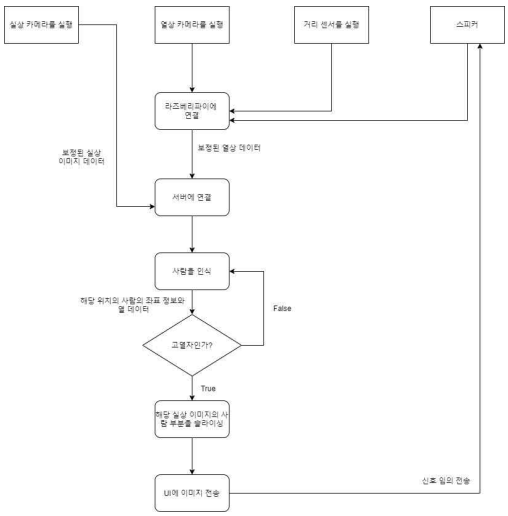


그림 7. 전체 프로세스 플로우 차트

① 열상 카메라와 스피커, 그리고 거리 센서를 라즈베리파이에 연결한다.
② 실상 카메라와, 라즈베리파이는 서버에 연결한다.
③ 실상 카메라는 실상 이미지 데이터를 서버에 송신한다.
④ 열상 카메라와 거리 센서를 이용해 온도를 보정한 값을 라즈베리파이가 처리하고, 서버에 송신한다.
⑤ 실상 카메라에서 받은 데이터를 가지고 사람을 Localization한다.
⑥ 열상 카메라에서 받은 좌표 데이터를 가지고, 사람의 좌표와 열의 데이터를 매칭시킨다.
⑦ 사람 좌표에 고열의 데이터(37도 이상)을 인식한 경우 고열자로 인식한다.
⑧ 고열자로 인식한 경우, 해당 사람 객체에 할당된 바운딩 박스의 좌표를 가지고 슬라이싱한다.
⑨ 슬라이싱한 이미지를 UI에 전송한다.

2.2 보정 기능

2.2.1 위치 보정 기능

- 열상 카메라와 실상 카메라는 서로 동일 렌즈를 사용하지 않기 때문에 서로 동일한 부분을 볼 수가 없다.
- 한 편 열상 카메라와 실상 카메라는 각각 화각이 다르므로, 보는 각도조차도 다르다.
- 이를 일치시키기 위해, 위치를 보정할 필요가 있다.
- 다음 논문을 참조하여 기능을 구현하였다. 김수창 외 1인, 2015, 한국정보통신학회논문지 2138-2144p, A Calibration Method for Multimodal dual Camera Environment

2.2.2.1 구현 방법

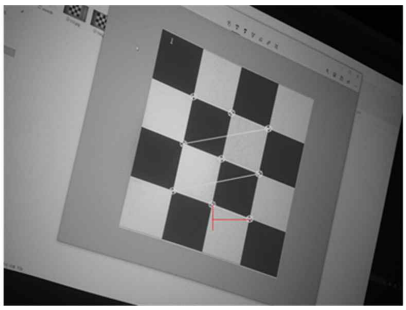


그림 8. 체크 보드의 점을 찍은 State

① 먼저 4 by 4의 체크보드를 생성한다.
② 열상 카메라와 실상 카메라에서 각각 체크보드 형태의 이미지를 얻는다.
③ 체크보드의 흰색과 검은색이 각각 만나는 점들을 찍는다.

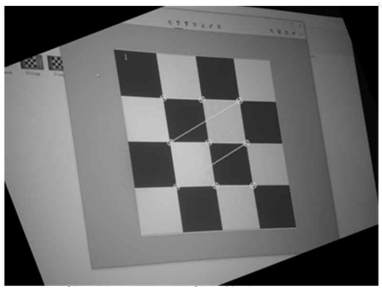


그림 9. 체크 보드를 회전

④ 찍힌 점들을 기준으로 하여 회전된 각도를 구한다.
⑤ 찍힌 점들 중(3, 3)에서, (2, 2)의 점을 기준을 중점에 두고 회전된 각도만큼 회전한다.

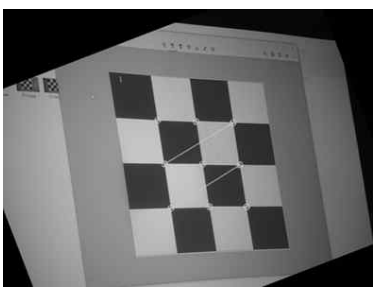


그림 10. 좌표 이동

⑥ 사전에 정의했던 (2, 2)을, 열상 카메라의 (2, 2)의 좌표와 비교하여 해당 지점으로 이동한다.
⑦ 체크 보드의 크기에 따라 배율을 조절한다.
- 위와 같이 테스트 케이스에 따라 조절한 경우, 회전 중점 (x, y) = (268.6204, 408.6848)이며, 회전된 각도는 21.48도가 나왔다. 실제로 값을 쟀을 땐, 21.03도가 나왔으며, 이는 2.09%의 오차를 의미한다.

2.2.2.2 코드

```
img = cv2.imread('res.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray = cv2.resize(gray, dsize=(640, 480), interpolation=cv2.INTER_AREA)
# Find the chess board corners

ret, corners = cv2.findChessboardCorners(gray, (3, 3), None)
#print("{}, {}".format(ret, corners))

if ret == True:
    objpoints.append(objp)
    cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
    imgpoints.append(corners)
    s = cv2.drawChessboardCorners(gray, (3, 3), corners, ret)
    #cv2.imwrite('stst.png', s)
    #cv2.imshow('KnV',gray)
    #cv2.waitKey(0)
cv2.destroyAllWindows()
```

그림 11. 위치 보정 용도 코드 1

- 다음은 위의 테스트 케이스를 구현하기 위해 작성된 코드이다.
- 먼저 해당 이미지를 읽는다. 그리고 체크 보드를 인식하기 편하도록, GRAY로 이미지를 변환한다. 그 다음, 4 by 4 케이스의 체크 보드이므로, findChessboardCorners 함수를 사용하여 점을 찾는다.
- 찾은 점을 s에 리스트로 넘긴다.

```python
if not corners[len(corners) - 1]:
    corners.pop()
# ================================================


# newX, newY에 좌표 x, y 배치 ====================
corners = list(map(float, corners))
newX = list()
newY = list()

for i in range(0, int(len(corners)/2)):
    newX.append(corners[2*i])
    newY.append(corners[2*i+1])

#print('x is {}\ny is {}'.format(newX, newY))
# ================================================


# 회전각 구하기 ==================================
print(newX[5], newY[5])
width = abs(newX[5] - newX[4])
height = abs(newY[5] - newY[4])
degree = math.atan(height/width) * 180 / 3.141592
print(degree) # 회전각
# ================================================



origin_height, origin_width = gray.shape[:2]
#M = np.float32([[1, 0, 100], [0, 1, 25]]) # right, left movement.
M = np.float32([[1, 0, 0], [0, 1, 0]]) # 변환 행렬, 좌측 우측
img_translation = cv2.warpAffine(gray, M, (origin_width, origin_height)) # img, convertArray, imageSize

# 이미지 회전
M = cv2.getRotationMatrix2D((268.6204, 408.68484), # 회전 중심, x, y
                            21.489730, # 회전각도(양수 반시계방향, 음수 시계방향)
                            1) # 이미지 배율
img_rotation = cv2.warpAffine(img_translation, M, (origin_width, origin_height))
cv2.imwrite('./res/res2.png', img_rotation)
```

그림 12. 위치 보정 용도 코드 2

- s의 값에 있는 x, y를 별도의 리스트인 newX, newY에 집어넣고, 이를 기반으로 회전각을 구한다.
- 그리고 어파인 함수를 통하여 위치를 이동하여 최종 보정한다.
- 위에 들어간 값들은, 값을 얻어서, 오른쪽에 직접 넣어서 보정한 것을 보이기 위해 작성된 것이다.
- 실제로 테스트 후 보정된 값은, 회전 각도 1.9697, 이미지 배율, 2.0407, 회전 중심은 (324.12906, 252.45726)이다.

2.2.2 온도 보정 기능

- 열상 카메라는 일반적으로 먼 거리일수록, 실제 온도와 인식된 온도의 차이가 발생하게 된다. 2m를 기준으로 최고 3도까지의 오차가 발생함을 확인하였다.
- 이를 보정하기 위해서는, 흑체를 사용하는 적외선 캘리브레이터를 사용해 복사 에너지를 계산하여 그 오차를 계산해 보정하는 것이 필수적이지만, 해당 캘리브레이터의 가격은 휴대용조차도 백 오십만원에 달하는 금액이어 보정할 수 없었다.
- 따라서 이를 최소한 방지하기 위해, 거리에 따라 달라지는 온도 차를 임의로 보정하는 것으로 해결하기로 하였다.
- 다음의 값은 거리에 따라 보정된 값이다.

거리(M)	보정 전(℃)	보정 후(℃)
0.1 ~ 0.2	36.0	36.03
0.2 ~ 0.3	35.19	35.90
0.3 ~ 0.4	34.45	35.48
0.4 ~ 0.5	34.17	35.49
0.5 ~ 0.6	33.93	35.16
0.6 ~ 0.7	33.49	34.91
0.7 ~ 0.8	33.23	35.80
0.8 ~ 0.9	32.95	35.01
0.9 ~ 1.0	32.19	35.13

표 4. 온도 보정 이후, 측정된 온도 값

- 1m를 기준으로, 0.1m마다 나누어 측정하였다. 20초 간 64fps마다 읽은 온도 데이터의 평균값으로 계산하였다. 즉, 한 케이스마다 1280개의 온도 데이터가 사용되었다.
- 실제 온도는 36.03도였으며, 가까울 수록, 보정전을 비교하면 일치함을 확인할 수 있다.
- 최대 오차율은 3.1%였으며, 최대 1.12도의 차이가 발생하였다.
- 하드웨어 스펙 상 1도의 차이가 발생하고 있는데다가, 1도의 차이가 발생한 경우가 한 경우에 치우쳐 있음을 감안하면, 오차 범위 내로 판단하기로 하였다.

2.3 탐지 기능

2.3.1. 설명

- 고열자를 인식하기 위해서는 먼저 사람을 인식할 필요가 있다.
- 사람을 인식하는 과정은 Joseph Redmon가 개발한 Yolo를 사용한다.
- Yolo는 one-stage-model로서, Detection과 Classification을 동시에 진행하기 때문에, 각각 따로 진행하는 Two-Stage-Model에 비해 훨신 빨라, 실시간 처리에 매우 유용하며, 이미 산업계 전반에 두루 사용된다.
- 웹캠을 통해 이미지를 수신하고, 해당 이미지 프레임에서 사람을 인식한다.
- 사람을 인식하고, 고열 데이터(37.5도 이상)의 데이터의 좌표와 연동하여, 해당 좌표와 사람의 좌표가 일치할 경우, 해당 사람은 고열자로 인식할 수 있다.
- 즉, 열상 카메라와 실상 카메라, 거리 센서를 종합하여 고열자를 판단하는, 센서 퓨전이다.
- 이를 순서도로 표현하면 다음과 같다.

2.3.1. 프로세스

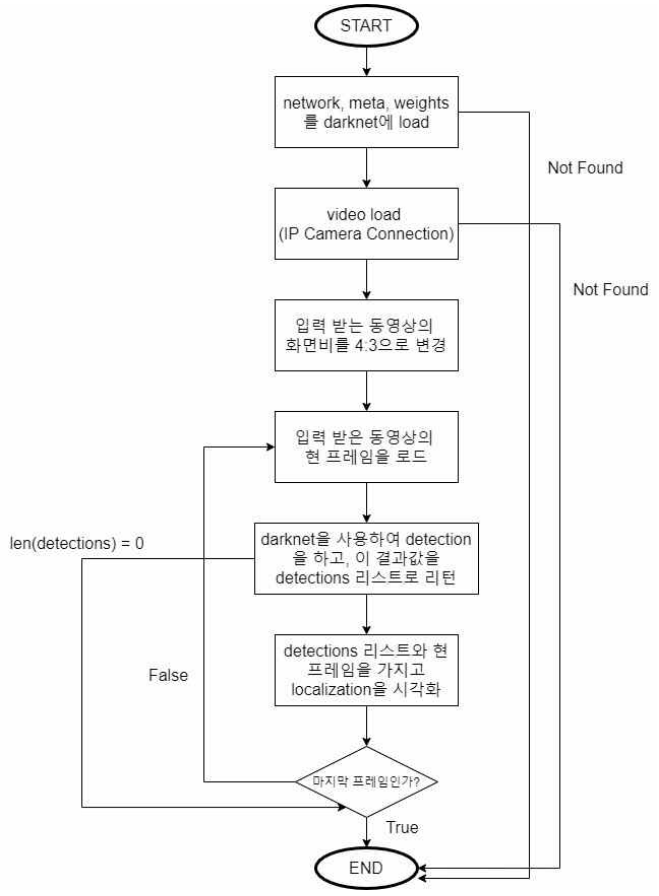


그림 13. 객체 탐지 프로세스 플로우 차트

- Darknet 작동을 위한 network(.cfg), meta(.data), weights(.weights)를 darknet에 로드한다
- 객체를 인식할 IP CAMERA와 연결한다
- 열화상 카메라의 비율이 4대 3이므로, 동기화를 위해 화면비를 4대 3으로 일치시킨다
- 입력 받은 비디오의 프레임 하나하나마다 darknet을 통해 detection을 수행하고 그 결과값을 detections 리스트로 리턴 받음. 이것을 가지고 localization을 시각화 한다.
- 이를 마지막 프레임까지 작동시킨다.
- 한 프레임을 기준으로, 앞서 객체 탐지에서 얻은 detections 리스트. 열화상 카메라에서 얻은 data를 리스트로 변환시켜 총 두 가지의 데이터를 획득한다
- 두 데이터를 동기화하기 위해 위치 보정 알고리즘으로 이를 보정한다.
- 실상 카메라 640 x 480 해상도를 기준으로 잡는다면, 열화상 카메라의 해상도가 32 x 24이므로, 20대 1의 비율로 데이터를 매칭시켜서, 센서 퓨전된 데이터를 리스트로 만들어낸다.
- 만든 데이터를 기반으로 하여, 사람이 인식된 부분에서 고열이 감지될 경우 고열자로 판단한다.

2.3.3 Yolo 소개 및 학습 초점

- Yolo는 Joseph Redmon가 2014년에 개발한, 객체 인식 모델이다. 상대적으로 인식률이 떨어지는 편이지만, 매우 빠른 속도로 인해 실시간 처리가 가능하다는 점으로 인해 산업계에 두루 사용된다.

2.3.3.1 왜 Yolo인가.

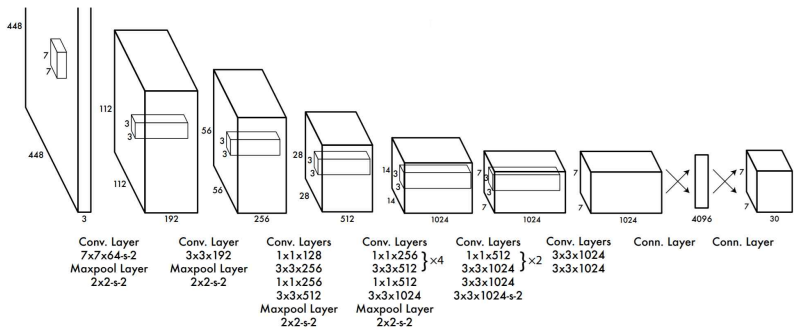


그림 14. Yolo v1의 네트워크 구조

- Yolo는 14년도 개발된 one-stage-model로 two-stage-model과는 차이가 있다.
- two-stage-model은 classification과 localization을 별도로 처리하지만, Yolo는 Grid를 그리고, 각 Grid마다 Classification과 Localization을 동시에 처리한다. 이 점에서 차이가 발생한다.
- 별도의 과정을 거칠 것 없이 단 한 번의 CNN을 통과하면 동시에 Classification과 Localization이 처리가 된다.
- 이는 곧 한 번에 처리를 하기 때문에 one-stage-model은 객체 인식 모델에서 속도에 특화되어있다. 이는 즉 매우 빠른 속도를 가지고 있음을 의미하고, 이는 실시간 처리가 용이함을 의미한다.

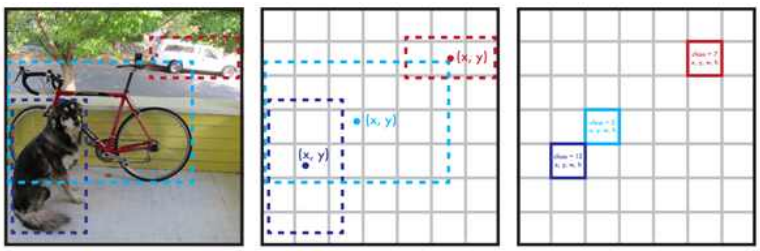


그림 15. Yolo v1의 Grid

- 주어진 이미지마다 Grid를 처리하고, 각 Grid마다, Localization할 Grid를 생성한다. 그리고 IOU (해당 논문에선 0.5 이상)인 경우 해당 바운딩 박스를 제거한다.
- Input Image를 NxN으로 구분하여 각 Grid Cell을 생성, Grid Cell마다 Region을 K-Mean Clustering을 이용하여 생성한다.
- Bounding Box마다 x, y, w, h, Confidence score 의 정보가 들어간다.

2.3.3.2 사용한 모델과 학습 방법.

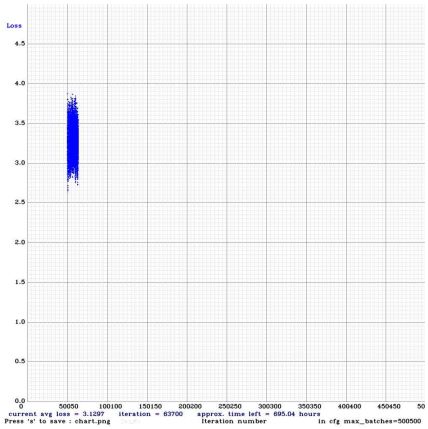


그림 16. 학습하여 생성된 Chart Image

- class는 person 단일
- Yolo v3를 사용했을 경우, 실시간 처리 프레임은 대략 20프레임 초중반대였으나, Yolo v4 모델로 전환 시 Yolo v4에서는 30대 중후반에서 40대의 프레임을 확보
- 다만 MOT 적용 시에, CPU를 주로 사용하는 MOT임에도 프레임이 6프레임대까지 떨어져 사용할 수 없어, 적용하지 아니함.
- 학습은 대략 6만 번까지 진행.

```
[net]
batch=64
subdivisions=32
# Training
#width=512
#height=512
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.00261
burn_in=1000
max_batches = 500500
policy=steps
steps=400000,450000
scales=.1,.1
```

그림 17. yolov4.cfg 파일 내의 hyperParameter

- 사용한 그래픽 카드의 1070ti의 메모리 용량이 8GB이므로, 개발용으로는 부족함. 최적의 hyperParameter를 찾아낼 필요가 있음.

YOLOv4: Optimal Speed and Accuracy of Object Detection

Alexey Bochkovskiy*
alexeyab84@gmail.com

Chien-Yao Wang*
Institute of Information Science
Academia Sinica, Taiwan
kinyiu@iis.sinica.edu.tw

Hong-Yuan Mark Liao
Institute of Information Science
Academia Sinica, Taiwan
liao@iis.sinica.edu.tw

Abstract

There are a huge number of features which are said to improve Convolutional Neural Network (CNN) accuracy. Practical testing of combinations of such features on large datasets, and theoretical justification of the result, is required. Some features operate on certain models exclusively and for certain problems exclusively, or only for small-scale datasets; while some features, such as batch-normalization and residual-connections, are applicable to the majority of models, tasks, and datasets. We assume that such universal features include Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial-training (SAT) and Mish-activation. We use new features: WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation, CmBN, DropBlock regularization, and CIoU loss, and com-

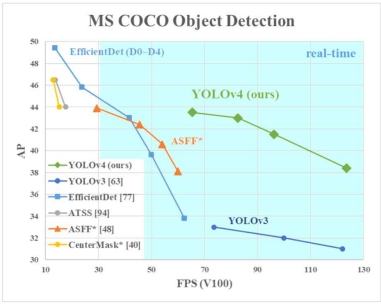


Figure 1: Comparison of the proposed YOLOv4 and other

그림 18. Yolo v4 Paper

- Yolo v3는 기존 Yolo에 대비하여 최적화에 집중한 모델이며, Yolo v4 또한 Genetic Algorithm을 통해 최적의 Parameter를 찾고, 기존 Yolo 모델들은 Grid 방식을 사용하기 때문에, 겹치거나 작은 객체에 대해서는 인식률이 매우 낮은데, 이를 input-resolution을 늘리는 것을 해결한다.
- 하지만 무엇보다 이 모델을 선택한 이유는 Mosaic Augmentation을 사용하면 4배 적은 Batch 사이즈로도 같은 학습 성능을 보인다.
- Self-Adversarial Training 란 방식을 제안하며, 이 것은 더 나은 정확도를 보여준다고 하지만, 이에 대한 테스트 결과는 없었음.
- 더 작은 하이퍼 파라미터, 더 빠른 성능은 기존 모델 대비하여, 프레임이 부족한 약점을 보완할 수 있었기 때문에 선택했다.
- 하지만 기존 v3는 6만번의 학습 이후에는 1.X에 달하는 Loss Function을 가졌지만, v4는 6만 번의 학습에도 불구하고 2.5X 대의 Loss Function을 보였다. 정확도를 높긴 하지만, 그에 비례하여 매우 많은 학습량이 요구되고 있었지만, 프레임이 필요했기 때문에 이를 선택했다.

2.4 경보 기능

- 라즈베리파이는 온도가 높은 픽셀의 데이터만을 서버에 전송한다. 따라서 경보 기능은 라즈베리파이에서 고열자 데이터가 전송된 경우에만 실행되고 있지만, 무인임을 감안하여 별도의 경보 기능을 추가하였다.
- TTS로 구현하고 싶었으나, 개발 시간 상 어렵다고 판단하였다.
- UI에서 버튼을 누를 경우, 다음과 같은 값이 라즈베리파이에 전송되고, 이에 따라 메세지가 스피커에서 음성으로 출력된다.

#1	잠시 멈춰주세요.
#2	담당자가 올 때까지 대기해주세요.
#3	뒤로 물러나주세요.
#4	간격을 두고 입장해주세요.
#5	일렬로 입장해주세요.

표 5. 문자열에 따라 스피커에서 발생하는 메세지

그림 19. 스피커로 사용하는 LG전자 PH1 무드

2.5 UI

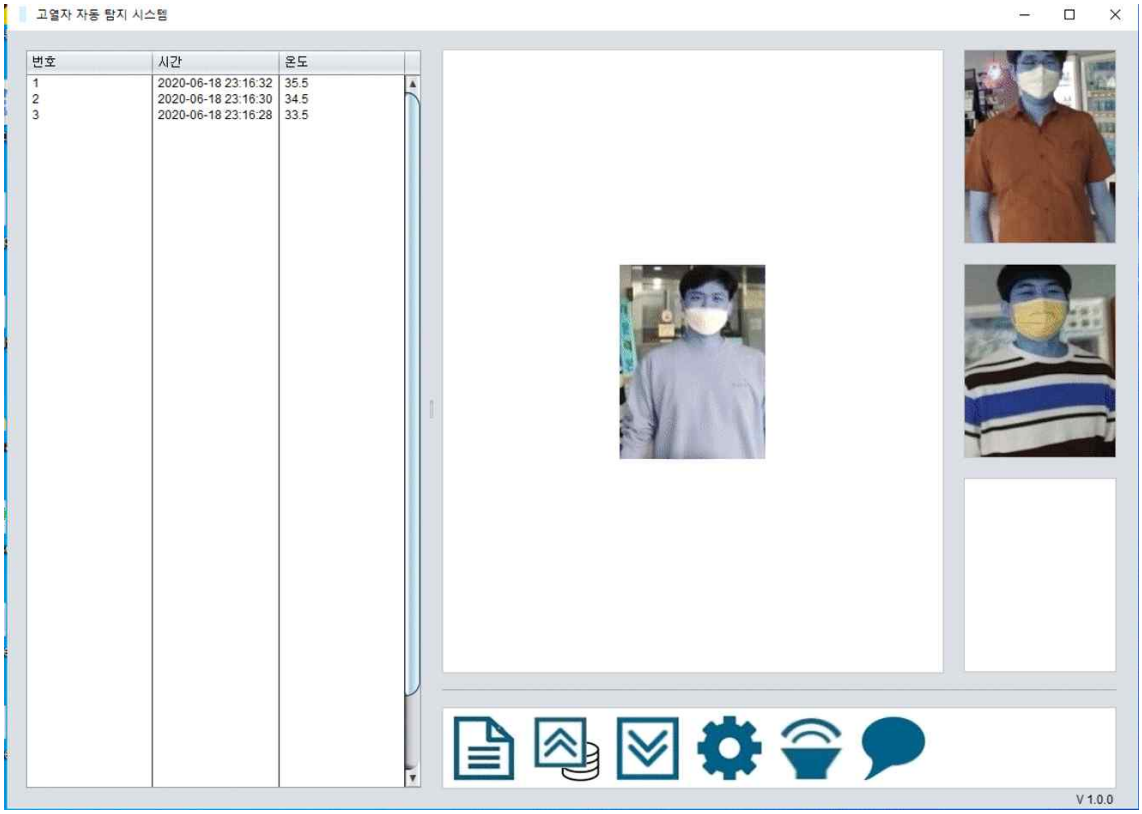


그림 20. UI

- UI 시스템은 서버에서 고열자를 인식하고, 해당 고열자를 슬라이싱한 사진을 UI에 전송한다. 가장 최신에 전송된 이미지가 우측의 가장 큰 박스 안에 위치되며, 이후 들어온 후에는 우측으로 차례대로 이동된다.

3. 구현

3.1 실상 카메라

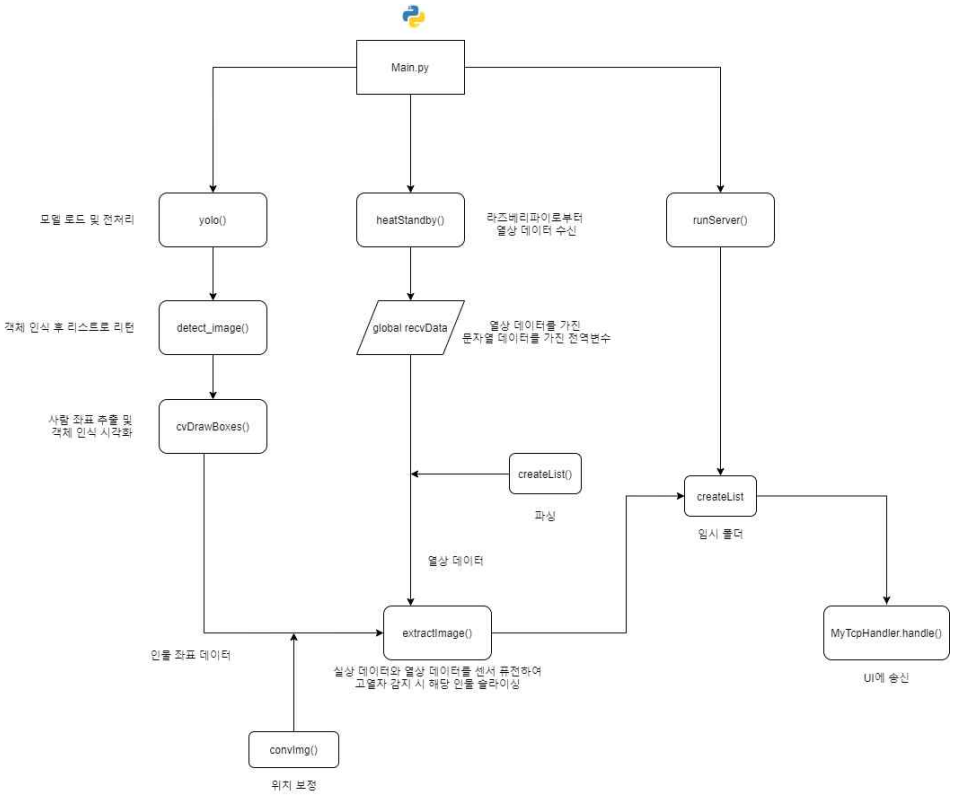


그림 21. 실상 카메라 프로세스 플로우 차트

① Main.py를 실행한다.
② yolo(), heatStandby(), runServer() 함수들을 스레드로 동시 동작한다.
③ yolo() 함수는 객체 인식 모델의 메타 데이터와 모델을 로드한다. 그리고 그 로드값에 기반해 dark_image란 해당 전처리된 모델을 생성한다.
④ dark_image 객체를 기반으로 detect_image 함수를 사용해 주어진 프레임에서 객체를 인식하고 img 리스트로 그 결과 값의 좌표 값을 리턴 받는다.
⑤ 그 값을 기반, 실상 데이터 값을으로, cvDrawBoxes()에 값을 넣는다. 이 후, 위치를 보정하고, extraImage() 함수를 실행해 센서 퓨징에 사용한다.
⑥ heatStandby() 함수는 라즈베리파이로부터 열상 데이터를 실시간으로 수신하여 recvData 전역 변수를 실시간으로 변화시킨다. 그리고 이 값을 파싱하여 열상 데이터로 바꾸고, extraImage에 넣는다. 이 때의 열 상 데이터가 센서 퓨징에 사용된다.
⑦ extraImage 함수에서 센서 퓨징되어 고열자를 인식하여 슬라이싱하고, runServer() 함수를 통해 UI에 송신한다.

3.1.1 기본 설정 및 모델 로드

■ main.py

```
import api, func, heatDetector
import threading, time, datetime
import os, sys
import serv
from socket import *



if __name__ == "__main__":
    yolo = threading.Thread(target=api.YOLO, args=())
    thermalVal = threading.Thread(target=heatDetector.heatStandby, args=())
    servImage = threading.Thread(target=serv.runServer, args=())

    yolo.start() # Yolo 가동
    thermalVal.start() # 실시간 열상 데이터 수신
    servImage.start()
```

그림 22. Main.py

- 실상 카메라에서 데이터를 추출하고, 보정하고, 슬라이싱하고, 센서 퓨전을 하는 서버의 메인 파일.
- ‘python main.py’ 로 실행한다.
- YOLO(), heatStandby(), runServer()를 스레드로 동시에 실행시킨다.
- YOLO() 함수는 Yolo 객체 인식을 실행시키는 함수, heatStandBy 함수는 열상 데이터를 실시간으로 받는 함수, runServer는 UI에 지속적으로 슬라이싱된 파일을 보내는 함수다.

■ api.py의 YOLO()

```
cap = cv2.VideoCapture("http://220.121.1.53:4747/video?dummy=param.mjpg")
cap.set(3, 1280)
cap.set(4, 720)


print("Starting the YOLO loop...")

# Create an image we reuse for each detect
darknet_image = darknet.make_image(darknet.network_width(netMain),
                                   darknet.network_height(netMain),3)

while True:
    prev_time = time.time() # prev_time, 프레임 계산하기 위한 처리 전 값.
    ret, frame_read = cap.read()

    frame_rgb = cv2.cvtColor(frame_read, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb,
                               (darknet.network_width(netMain),
                                darknet.network_height(netMain)),
                               interpolation=cv2.INTER_LINEAR)

    darknet.copy_image_from_bytes(darknet_image,frame_resized.tobytes())
    detections = darknet.detect_image(netMain, metaMain, darknet_image, thresh=0.8)
    image = func.cvDrawBoxes(detections, frame_resized, frame_rgb)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    print(1/(time.time()-prev_time)) # 현 시간 - 앞서 처리 전 시간 = 프레임 출력

    '''
    os.system('clear')
    print(1/(time.time()-prev_time)) # 현 시간 - 앞서 처리 전 시간 = 프레임 출력
    if func.feverSound == True:
        print('========================')
        print('DETECT THE HIGH FEVER')
        print('========================')
        func.feverSound = False
    '''


    cv2.imshow('Demo', image) # 현 프레임 이미지 출력
    cv2.waitKey(3)
cap.release()
out.release()
```

그림 23. api.py

- 위에서 모델을 로드하고, 웹캠에 연결한다.
- frame을 객체 인식하기 위해 사전 전처리하고, 학습된 모델에 따라 객체 인식한다
- 객체 인식된 값의 리스트를 detections 리스트에 넣는다.
- 이후 고열자를 인식하고, 바운딩 박스를 그릴 수 있는 cvDrawBoxes 함수에 넣는다.

3.1.2 객체 검출 및 센서 퓨전

■ func의 cvDrawBoxes()

```
def cvDrawBoxes(detections, img, origin):
    global a, b, c, d, num
    for detection in detections:
        x, y, w, h = detection[2][0],\
            detection[2][1],\
            detection[2][2],\
            detection[2][3]
        xmin, ymin, xmax, ymax = convertBack(
            float(x), float(y), float(w), float(h))

        pt1 = (xmin, ymin) # 좌상단
        pt2 = (xmax, ymax) # 우하단
        path = '/home/inha/다운로드/space_origin/'

        a, b, c, d = returnPointer(xmin, ymin, xmax, ymax)

        origin = cv2.cvtColor(origin, cv2.COLOR_BGR2RGB)
        cv2.imwrite(path + timeName() + '.jpg', origin) # originFrame
        extractImage(img, xmin, xmax, ymin, ymax, origin) # frameSlicing
        cv2.rectangle(img, pt1, pt2, (0, 255, 0), 1)


    return img
```

그림 24. func.py

- 객체를 검출하고 슬라이싱하는 함수.
- 객체를 인식하여 해당 좌표 값을 받은 리스트 detections, 객체 인식을 하기 위해 사용되었던 전처리된 프레임 img, 그리고 원본 프레임 origin을 매개변수로 넘긴다.
- 원본 프레임을 매개 변수로 넘기는 것은 객체 인식을 하기 위해 사용되었던 프레임은 변환되었기 때문에 보기에 좋지 않도록 손상되어 있기 때문이다. 해당 이미지에서 데이터를 뽑아내고, 원본 프레임에 다시 재변환하여 슬라이싱한다.
- Yolo는 기본적으로 좌상단의 좌표와 높이, 너비 값을 리턴한다. 이를 좌표로 변환하기 위해 convertBack()를 이용하여, 객체의 좌상단 좌표, 우상단 좌표를 얻는다.
- 좌표를 얻었으면, 원본 프레임, 전처리된 프레임, 그리고 각 좌표값을 extractImage 함수에 전달한다.
- rectangle() 함수를 이용하여, 인식된 객체마다 바운딩 박스를 그려 시각화한다.
- 바운딩 박스가 그려진 이미지를 리턴한다.

■ func의 extractImage()

```python
def extractImage(img, x1, x2, y1, y2, origin):
    global num
    global feverSound

    if x1 < 0 or x2 < 0 or y1 < 0 or y2 < 0:
        return -1 # 데이터가 없을 시 그냥 종료

    x1 = int(x1)
    x2 = int(x2)
    y1 = int(y1)
    y2 = int(y2)

    path = '/home/inha/다운로드/space/'

    # 실상 이미지 데이터 보정
    img = convImg(img)

    # 열상 이미지 데이터

    accords = heatDetector.recvData
    if accords == '0,0':
        return
    accords = accords.split('|')
    a, b = createList(accords)

    chkPerson = False
    num += 1
    for i in range(0, len(a)):
        x = a[i]
        y = b[i]
        if x1 <= x and x <= x2 and y1 <= y and y <= y2:
            chkPerson = True
            break
        else:
            pass

    # 잘못인식된 경우 제외
    if x1 <= x and x <= x2 and y1 <= y and y <= y2:
        pass
    else:
        return # 해당 좌표 위치에 없을 시 종료


    # FHD 해상도 전용
    '''
    cp = origin.copy()
    x1 = int(4.615*x1)
    x2 = int(4.615*x2)
    y1 = int(2.596*y1)
    y2 = int(2.596*y2)
    '''

    # SD 해상도 전용
    cp = origin.copy()
    x1 = int(1.538*x1)
    x2 = int(1.538*x2)
    y1 = int(1.153*y1)
    y2 = int(1.153*y2)


    # 고열자 있을 시 해당 사진 슬라이싱
    if chkPerson: # 고열자가 있었다면
        feverSound = True
        cp = cp[y1:y2, x1:x2]
        try:
            cv2.imwrite(path + str(num) + '.jpg', cp)
        except:
            return
    else:
        return
```

그림 25. func의 extraImage()

- 객체가 인식되고, 고열자임이 판단된 경우 슬라이싱하는 함수
- 객체가 인식되지 않은 경우 슬라이싱을 중지한다.
- 슬라이싱 하기 전에, 이미지를 먼저 보정한다.
- 앞서 heatStandby에서 얻은 열상 이미지 데이터를 파싱하여 얻어내고, 해당 사람이 인식된 객체의 좌표 내에 고열이 있는지 판단한다. 고열이 있다면, 고열자로 판단한다.
- 해상도에 맞추어 좌표를 변경하고, 고열자라 판단된 객체를 슬라이싱한다. 해상도에 맞추어 바꾸어 좌표를 변경한다. 이는 전처리 된 프레임으로 슬라이싱되면 이미지가 시각적으로 손상되어 있기 때문이다. 이를 방지하기 위해 원본 프레임으로 값을 변경하여 슬라이싱한다.
- 이렇게 실상 데이터와 열상 데이터를 가지고 센서 퓨전하고 고열자를 판단한다.

3.2 열상 카메라

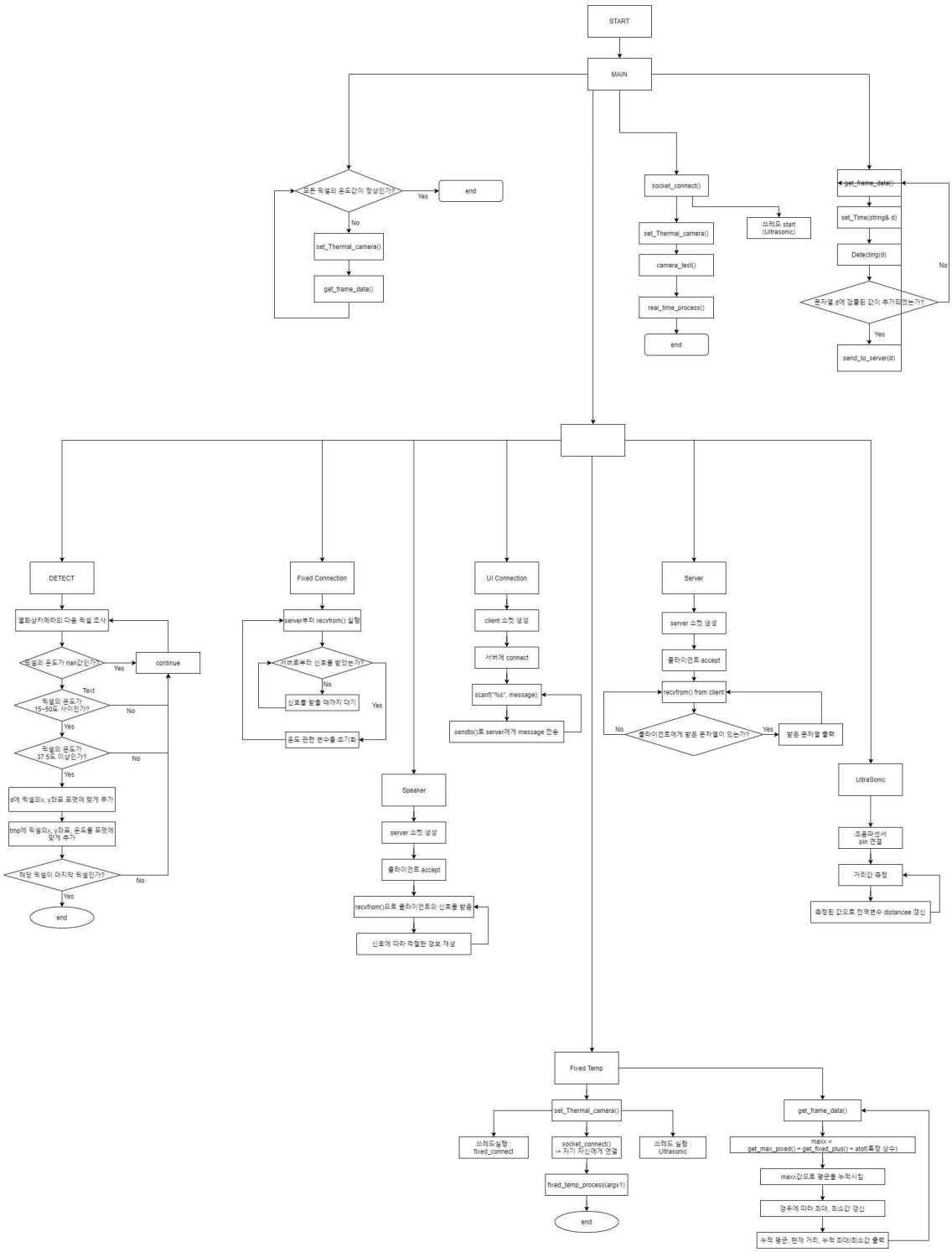


그림 26. 열상 카메라의 프로세스 플로우 차트

- 열상 카메라는 36 x 24의 resoultion을 갖고 있으며, 오차 ±1° 이다. 이 값을 서버에 지속적으로 송신하여 고열자를 판단하는 데이터로 활용된다.
- UI에서 임의의 값을 받을 경우 스피커를 통해 메세지를 송출한다.

3.1.1 Total_test.cpp

- 주요 메인 함수. 열상 데이터를 인식하고 송신한다.

```
void Ultrasonic() { //쓰레드 함수 : 초음파 센서
                    //delay 마다 전역변수 거리값 distancee를 센서에서 읽어와서 갱신함
    long start, stop;
    if (wiringPiSetup() == -1) { printf("wiringPiSetup() error\n"); exit(0); }

    pinMode(GPIO_TRIGGER, OUTPUT);  //pin 연결설정
    pinMode(GPIO_ECHO, INPUT);

    while (1)   //delay마다 거리값 갱신
    {
        digitalWrite(GPIO_TRIGGER, LOW);
        delay(50);
        digitalWrite(GPIO_TRIGGER, HIGH);
        delayMicroseconds(10);
        digitalWrite(GPIO_TRIGGER, LOW);
        while (digitalRead(GPIO_ECHO) == 0);
        start = micros();
        while (digitalRead(GPIO_ECHO) == 1);
        stop = micros();

        float tmp = (float)(stop - start) / 58.8235;

        if (tmp > 0 && tmp <= 200)
            distancee = tmp;
        delay(50);
    }

    return;
}
```

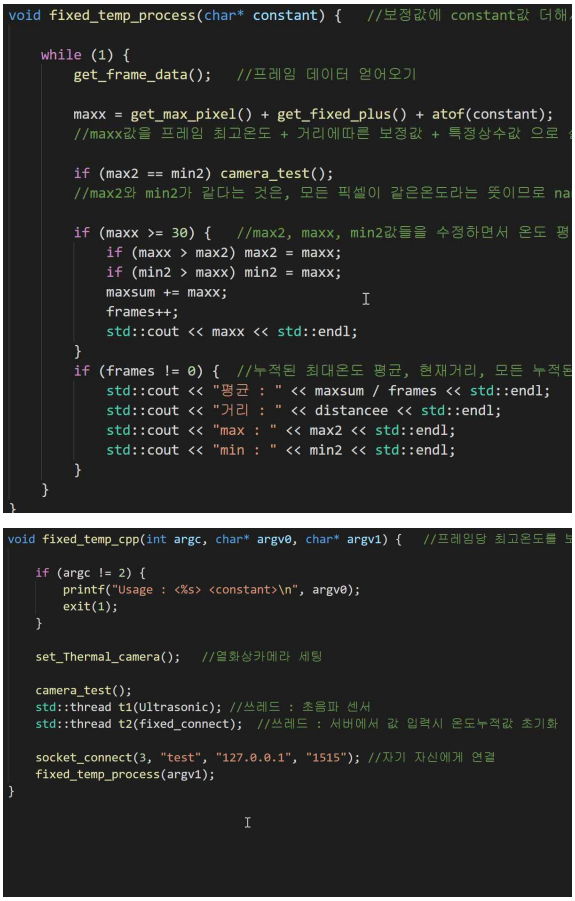
그림 27. Total_test.cpp의 Ultrasonic()

- UltraSonic() 함수는 초음파 센서에서 신호를 받고, distancee 전역 변수에 실시간으로 값을 업데이트한다. 이 값을 통해 열상 데이터를 보정하는데 사용된다.

```
void get_frame_data() { //프레임 픽셀 온도데이터 얻어오기 -> mlx90640To[]에 저장
                        //한 프레임의 온도데이터를 전역변수 float mlx90640To[768]에 저장합니다.
    state = !state;
    MLX90640_GetFrameData(MLX_I2C_ADDR, frame);
    eTa = MLX90640_GetTa(frame, &mlx90640);
    subpage = MLX90640_GetSubPageNumber(frame);
    MLX90640_CalculateTo(frame, &mlx90640, emissivity, eTa, mlx90640To);
    MLX90640_BadPixelsCorrection((&mlx90640)->brokenPixels, mlx90640To, 1, &mlx90640);
    MLX90640_BadPixelsCorrection((&mlx90640)->outlierPixels, mlx90640To, 1, &mlx90640);
}
```

그림 28. Total_test.cpp의 get_frame_data()

- get_frame_data() 함수는 열상 데이터 값을 받아오고, mlx90640To 배열에 전달한다.



```
void fixed_temp_process(char* constant) {   //보정값에 constant값 더해
    while (1) {
        get_frame_data();   //프레임 데이터 얻어오기

        maxx = get_max_pixel() + get_fixed_plus() + atof(constant);
        //maxx값을 프레임 최고온도 + 거리에따른 보정값 + 특정상수값 으로

        if (max2 == min2) camera_test();
        //max2와 min2가 같다는 것은, 모든 픽셀이 같은온도라는 뜻이므로 na

        if (maxx >= 30) {   //max2, maxx, min2값들을 수정하면서 온도 평
            if (maxx > max2) max2 = maxx;
            if (min2 > maxx) min2 = maxx;
            maxsum += maxx;
            frames++;
            std::cout << maxx << std::endl;
        }
        if (frames != 0) {  //누적된 최대온도 평균, 현재거리, 모든 누적된
            std::cout << "평균 : " << maxsum / frames << std::endl;
            std::cout << "거리 : " << distancee << std::endl;
            std::cout << "max : " << max2 << std::endl;
            std::cout << "min : " << min2 << std::endl;
        }
    }
}
```

```
void fixed_temp_cpp(int argc, char* argv0, char* argv1) {   //프레임당 최고온도를 보

    if (argc != 2) {
        printf("Usage : <%s> <constant>\n", argv0);
        exit(1);
    }

    set_Thermal_camera();   //열화상카메라 세팅

    camera_test();
    std::thread t1(Ultrasonic); //쓰레드 : 초음파 센서
    std::thread t2(fixed_connect);  //쓰레드 : 서버에서 값 입력시 온도누적값 초기화

    socket_connect(3, "test", "127.0.0.1", "1515"); //자기 자신에게 연결
    fixed_temp_process(argv1);
}
```

그림 29. func의 fixed_temp_process(), fixed_temp_cpp()

- fixed_temp_process() 함수와 fixed_temp_cpp() 함수는 최종적으로 얻은 거리와 열상 데이터를 통해 최종적으로 값을 출력한다.

3.2.2. UClient.cpp

```
void error_handling(char *message);

int sock;
struct sockaddr_in serv_adr;
socklen_t serv_adr_sz;

int main(int argc, char* argv[]) {

    ////소켓 생성 및 연결
    serv_adr_sz = sizeof(serv_adr);

    if (argc != 3) {
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);
    if (sock == -1)
        error_handling("socket() error");

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family = AF_INET;
    serv_adr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_adr.sin_port = htons(atoi(argv[2]));

    if (connect(sock, (struct sockaddr*) & serv_adr, sizeof(se
        error_handling("connect() error!");
    }
    else
        puts("Socket Connected.....");
    ////소켓 생성 및 연결 완료


    char* message[100];
    while (1) {         //고열자가 감지되면, 라즈베리파이 쪽에 분

        scanf("%s", message);   //임의로 키보드입력을 받아서 고열
        sendto(sock, message, BUF_SIZE, 0,
            (struct sockaddr*) & serv_adr, sizeof(serv_adr));
    }
    close(sock);
    return 0;
```

그림 30. UClient.cpp

- UI와 연결 한다.
- 이후, 고열자 감지 이후, 신호를 연동한다.

3.2.3 real_speaker.cpp

```
char message[500];
int read_cnt;

while (1) {     //신호를 받았을 시, 경보를 재생하고, 화면에 신호를 받았다고 출력합니다.
                //또한, 받은 신호에 따라 여러가지 경보음을 울립니다.
    read_cnt = recvfrom(clnt_sock, message, 500, 0,
        (struct sockaddr*) &clnt_adr, &clnt_adr_sz);
    if (read_cnt > 0) {
        printf("\n\n\n\n\nSignal received!\n");
        printf("\n\n\n\n\n\n\n\n\n\n\n\n\n");
        if (strcmp(message, "#1") == 0) system(MP3_PLAY_CMD1);
        if (strcmp(message, "#2") == 0) system(MP3_PLAY_CMD2);
        if (strcmp(message, "#3") == 0) system(MP3_PLAY_CMD3);
        if (strcmp(message, "#4") == 0) system(MP3_PLAY_CMD4);
        if (strcmp(message, "#5") == 0) system(MP3_PLAY_CMD5);
        printf("received message : %s\n", message);
    }

    char tmp_char;
    long tmp_long = 0;

    if (ioctl(clnt_sock, FIONREAD, &tmp_long) != -1) { //소켓에 쌓인 나머지 값을 버림
        for (int i = 0; i < tmp_long; i++)
            recv(clnt_sock, &tmp_char, sizeof(char), 0);
    }
}

close(clnt_sock);
close(serv_sock);
return 0;
```

그림 31. real_speaker.cpp

- 신호를 받으면, 무인 상태에서, 메세지를 스피커를 통해 송출하는 파일
- 블루투스 스피커에 연동한 후, 출력한다.

3.3 UI

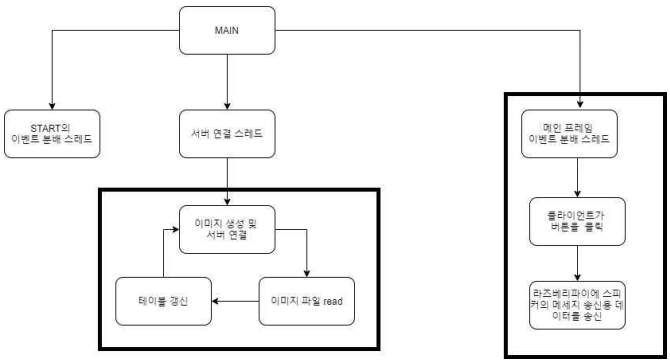


그림 32. UI 프로세스 플로우 차트

- 서버에서 생성한 고열자 데이터를 수신하는 역할을 담당한다.
- 라즈베리파이에 임의로 원하는 신호를 송신하는 역할을 담당한다.

3.3.1 사진 출력

```
if(side_label1.getIcon() != null && side_label2.getIcon() != null
        && side_label3.getIcon() != null && center_label.getIcon() != null) {
    side_label3.setIcon(side_label2.getIcon());
    side_label2.setIcon(side_label1.getIcon());
    side_label1.setIcon(center_label.getIcon());
    center_label.setIcon(new ImageIcon(str));
}

if(center_label.getIcon() == null) {
    center_label.setIcon(new ImageIcon(str));
}
else if(side_label1.getIcon() == null) {

    side_label1.setIcon(center_label.getIcon());
    center_label.setIcon(new ImageIcon(str));
}
else if(side_label2.getIcon() == null) {

    side_label2.setIcon(side_label1.getIcon());
    side_label1.setIcon(center_label.getIcon());
    center_label.setIcon(new ImageIcon(str));
}
else if(side_label3.getIcon() == null){

    side_label3.setIcon(side_label2.getIcon());
    side_label2.setIcon(side_label1.getIcon());
    side_label1.setIcon(center_label.getIcon());
    center_label.setIcon(new ImageIcon(str));
}

for(int i=check; i>0; i--) {

    ob[i][0] = Integer.valueOf(ob[i-1][0].toString())+1;;
    ob[i][1] = ob[i-1][1];

}
```

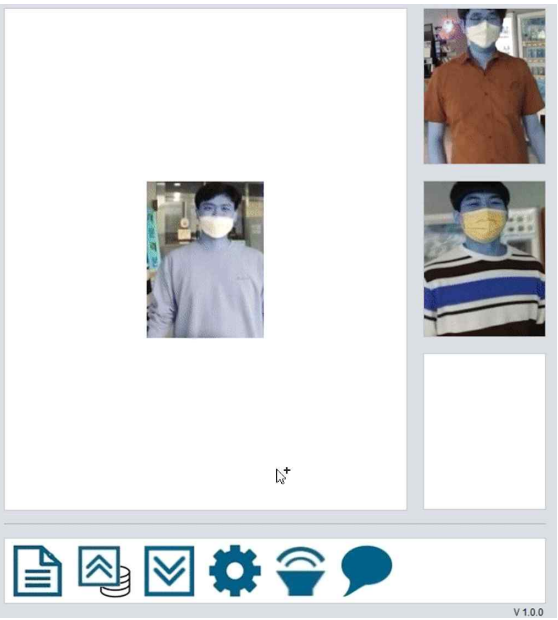


그림 32. 사진 출력 순서

- 이미지를 수신 받으면, 각각 차례대로 우측에 배치한다.
- 가장 최신을 좌측, 그 이후는 우측 상단부터 차곡차곡 배치한다.

3.3.2 신호 송신

```
button1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Socket soc1;
        try {
            soc1 = new Socket("220.121.1.53", 1516);
            BufferedOutputStream out;
            out = new BufferedOutputStream(soc1.getOutputStream());
            out.write(1);
            out.flush();

            soc1.close();
            out.close();

        } catch (UnknownHostException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

});
```

그림 33. 신호 송신 파트

- 버튼을 누르면, 신호 1을 라즈베리파이에 보낸다. 그러면 임의의 메세지가 라즈베리파이에 연결된 스피커에서 메세지가 출력된다.
- 이하, 마찬가지로 신호를 추가하여 여러가지 메세지를 출력한다.

4. 결론

작동 영상은 다음 동영상 URL을 참조하십시오.