
영상분석 인수인계서

이동원 연구원 인수인계 보고서

목차

영상분석 메인 소스코드 (X_Parking_inout)	7
1. 개요	7
2. Yolov5 API 소스코드 설명	8
A. data 디렉토리	8
B. hyps 디렉토리	8
C. images 디렉토리	8
D. scripts 디렉토리	8
E. models 디렉토리	8
F. utils 디렉토리	8
a. general.py	8
b. plots.py	9
c. detect.py	9
d. export.py	9
e. train.py	9
3. 주차 현황 서비스용 소스코드 설명	10
A. config *.yaml	10
a. src	10
b. pos	10
c. uuid	10
d. roi	10
e. loi	10
f. double_rois (optional)	10
g. double_slots (optional)	10
B. check_state.py	11
a. inout	11
b. occupancy_inout	12
c. find_intersection	15
d. trace_inout	18
C. dataloader.py	21
D. detector.py	21
a. __make_img	21
b. detect	21
E. run_final.py	22
a. main	22
b. insert_data (optional)	23
F. transmit_server.py	23
a. sendstate2server	23
b. put, get	25
G. verify.py	25

4.	영상분석 서비스 실행 방법	27
A.	서버 OS 설치 및 환경설정	27
B.	config_*.yaml 파일 작성	27
a.	ROI / LOI 설정	27
b.	slot id, uuid 입력	28
c.	카메라의 RTSP 스트리밍 주소 입력	28
d.	road 값 입력 (optional)	28
e.	double_roi (optional)	28
f.	double_pos (optional)	28
C.	run_final.py 실행 매개변수 설정	29
a.	--weights	29
b.	--conf-thres	29
c.	--iou-thres	29
d.	--device	29
e.	--exist-ok	29
f.	--line-thickness	30
g.	--send-mqtt	30
h.	--service	30
i.	--config-file	30
j.	--mode (optional)	30
D.	작업 스케줄러를 이용한 자동 실행 스크립트 작성	30
a.	runCheck.sh	30
b.	작업 스케줄러 등록	32
E.	프로세스 실행 여부 확인	33
5.	MQTT 를 이용한 영상분석 결과 데이터 수집 (Optional)	34
A.	mqtt 디렉토리	34
a.	MQTTConfigBuilder.py	34
b.	Registration_MQTT_with_pem_key.py	35
c.	Login_MQTT.py	37
d.	Service_MQTT.py	37
B.	run_final.py 에 적용	37
	실내 주차장 내 보행자 인식 (Pedestrian2Map)	40
1.	개요	40
A.	개발 환경	40
B.	Homography	41
2.	보행자 인식 소스코드 설명	42
A.	util/config_hd_2.py	42
B.	main_seq.py	42
a.	getFrame	42
b.	detect	43

c. 조건별 성능 분석.....	45
C. <code>test_save_photo.py</code> - <code>getframe</code>	46
3. 고찰.....	47
4. 실행 방법.....	48
실시간 장애물 위치 정보 제공 툴 (ClickPedestrian).....	49
1. 개요.....	49
2. 필수 라이브러리.....	49
3. 실행 방법.....	49
실내 주차장 내 차량 Tracking (Car_Tracking).....	50
1. 개요.....	50
2. stitching 알고리즘	50
A. 추가 사항.....	50
3. 실행 방법	52
4. <code>main_seq.py</code>	52
5. 실행 결과.....	52
6. 문제점	53
친환경 차량 검출 및 번호판 OCR (ALPR).....	54
1. 개요.....	54
2. 개발 환경.....	54
3. util 디렉토리.....	56
A. <code>KakaoOCR.py</code>	56
B. <code>tesseract_OCR.py</code>	56
C. <code>Easy_OCR.py</code>	56
4. YOLO 학습	57
5. HSV Masking.....	58
6. 실행 결과.....	59

부록	61
1. 영상분석 서버 내 운영체제 설치 및 환경설정	61
A. <i>Proxmox 가상화 하이퍼바이저 OS 설치</i>	61
a. Proxmox OS 설치 이미지 다운로드	61
b. 호스트 시스템 설치.....	61
c. 게스트 시스템 설치.....	66
B. <i>GPU Passthrough 활성화</i>	68
C. <i>Anaconda 가상환경 설치 (Optional)</i>	70
a. Anaconda 설치 파일 다운 및 실행	71
b. 가상 환경 생성	71
D. <i>NVIDIA CUDA Toolkit, CUDNN 설치</i>	71
a. Debian 설치 패키지 파일 다운로드	72
b. CUDA 공개 GPG 키 추가	72
c. CUDA 설치	72
d. 시스템 환경변수 추가	72
e. CUDNN 설치	73
f. 설치 확인	73
E. <i>OpenCV 설치</i>	73
a. pip 를 이용한 설치	73
b. 사용자 지정 옵션을 이용한 설치	74
F. <i>PyTorch 및 YoLov5 필수 라이브러리 설치</i>	76
a. PyTorch 설치 파일 다운로드 및 설치	76
2. 영상분석 서버 접속 IP 리스트 및 계정 정보.....	78
3. 영상분석용 주차장 CCTV RTSP 접속 정보.....	79
4. 주차장 수동 관제 모드 실행방법.....	79
5. 디스크 파티션 공간 조정 방법 (리눅스 기준).....	81
A. <i>Proxmox 호스트 웹페이지에서 디스크 용량 증가</i>	81
B. <i>게스트 시스템 내 파티션 용량 증가</i>	81
6. NVIDIA Jetson Nano 설치 및 환경설정	83
A. <i>JetPack 설치</i>	83
B. <i>jetson-stats 설치</i>	84

C.	<i>gdm3 삭제 및 lightdm 설치</i>	84
D.	<i>영상분석용 라이브러리 설치</i>	84
a.	OpenCV 4.5.4 with CUDA.....	84
b.	PyTorch 1.8 + torchvision v0.9.0 다운로드.....	85
c.	Cython, numpy, pytorch 패키지 설치	85
d.	torchvision 패키지 설치.....	85
e.	Yolov5 실행용 필수 패키지 설치	86
f.	Yolov5 소스코드 사용	86
7.	GitLab 프로젝트 리스트	87

영상분석 메인 소스코드 (X_Parking_inout)

1. 개요

주차장에서 사용하는 영상분석 소스코드는 YOLOv5

(<https://github.com/ultralytics/yolov5>)를 기반으로 개발하였다. YOLOv5는 기본적으로 차량 인식은 가능하나, 카메라의 화각에 따라 인식 정확도가 크게 변화하여 주차장 내 차량 이미지를 라벨링하여 새로운 YOLOv5 모델을 학습시키는 것이 좋다. YOLOv5의 소스코드는 Python3를 기반으로 개발되었으며, 시간이 지나면서 소스코드의 API가 업데이트되고, 2022년 9월 현재 API 버전 7.0까지 출시된 상태이다. 현재 주차장에 적용중인 YOLOv5 소스코드의 API 버전은 주로 버전 6.0/6.1에 기반을 두고 있으나, 주차장의 규모에 따라 5.0 버전이 적용되기도 한다. 버전 6.0과 5.0의 가장 큰 차이는 영상 로딩 시 비동기 스레드의 적용 유무이며, 이를 사용하면 영상분석 수행 시간을 줄일 수 있는 장점이 있으나, CPU 사용량이 늘어나기 때문에 CPU 사양에 맞게 사용 여부를 결정해야 한다. 그리고 서비스 고도화를 위해 추가적인 알고리즘이 적용되어 있으며, 주차장마다 적용된 알고리즘이 다소 상이하므로 [표 1]과 같이 정리했다. 각 알고리즘에 대한 설명은 다음 항목으로 이어진다.

주차장명	서비스 지역	YOLOv5 API 버전	주차 판단	데이터 수집 (Y/N)	이중주차 처리 (Y/N)	비고
안양 2차 SKV1	지하 3층	6.0	ROI	Y		
천호역 공영주차장	지하 전층	5.0	ROI	N	N	
강동역 공영주차장	지하 전층	6.0	ROI	Y	N	천호역과 연결
호반파크 2관	지하 3,4층	6.0	ROI	Y	N	9월 중 서비스종료
안산 상하수도 공영주차장	야외 전 구역	6.1	LOI	N	N	9월 중 서비스오픈
안산 그랑시티자이	지하 전층				Y	개발예정
안양 공영주차장	지상, 옥상 전층				N	개발예정

표 1. 영상분석 서비스 적용 주차장 목록

2. YOLOv5 API 소스코드 설명

이 항목에서는 YOLOv5의 Github 프로젝트

(<https://github.com/ultralytics/yolov5>)에 포함된 파일 및 디렉토리에 대한 간략한 설명을 포함한다. 이 파일들은 YOLOv5 모델 구현에 필요한 세부 정보를 포함하고 있으므로, 해당 파일들의 역할에 대하여 명확히 이해한 후 수정 또는 삭제하는 것을 권장한다. 여기에서 설명하지 않은 파일에 대한 설명은 프로젝트 홈페이지 (<https://github.com/ultralytics/yolov5>)를 참고하기 바란다.

A. `data` 디렉토리

Object Detection을 위한 데이터셋의 설정 값이 저장된 `yaml` 파일이 있는 디렉토리이다. 실행 시 별도의 매개변수를 지정하지 않으면 기본값으로 `coco.yaml` 파일을 로딩하며, 이 파일에는 coco 데이터셋의 경로, 학습/검증/테스트 데이터셋의 레이블 정보, 클래스 이름 (80 가지)이 기록되어 있다.

B. `hyp` 디렉토리

모델 학습 시 필요한 하이퍼 매개변수 (hyperparameter)의 값이 저장된 디렉토리

C. `images` 디렉토리

예제 이미지가 저장된 디렉토리

D. `scripts` 디렉토리

YOLOv5 모델 기본 가중치 파일 (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x) 다운로드 셸 스크립트 파일 (`download_weights.sh`), coco / coco128 / imagenet 데이터셋 다운로드 셸 스크립트 파일 (`get_coco.sh` / `get_coco128.sh` / `get_imagenet.sh`)이 포함된 디렉토리

E. `models` 디렉토리

YOLOv5 모델 구조 정보를 포함하는 디렉토리

F. `utils` 디렉토리

YOLOv5 모델 실행에 필요한 파일을 포함하는 디렉토리

a. `general.py`

105 번째 줄부터 정의된 `set_logging` 함수는 모델 실행 결과를 출력하는 로그 모듈 및 출력 형식을 설정한다. 114 번째 줄 `logging.Formatter` 함수의 인자로 출력 형식을 지정할 수 있다. 예를 들어, `'%(asctime)s - %(name)s - %(levelname)s - %(message)s'`를 인자로 사용하고 실행하면 콘솔 창에 [표 2]와 같이 출력된다.

자세한 정보는 Python 공식 홈페이지 내 로깅 문서

(<https://docs.python.org/ko/3/howto/logging.html>)를 참고하기 바란다.


```
2022-08-17 10:00:27,816 - root - INFO - B3F-09 - Parking state w/o  
freezing: {'3-y5-x2-1': 'in', '3-y5-x2-2': 'in', '3-y5-x2-3': 'in', '3-  
y5-x3-1': 'in', '3-y5-x3-2': 'in'}
```

표 2. 로그 메시지 출력 예시

395 번째 줄부터 정의된 `check_imshow` 함수는 영상분석 전 이미지/영상 출력 가능여부를 테스트하는 함수이다. GUI 환경에서 소스코드 실행 시 작은 창이 잠깐 나타났다가 사라지는데, CLI 환경에서는 오류를 발생시키며 영상분석이 시작되지 않기 때문에 `cv2.imshow('test', np.zeros((1, 1, 3)))` 부분부터 `cv2.waitKey(1)` 부분까지 4 줄을 주석 처리하거나 메인 코드에서 `check_imshow` 함수를 호출하지 않도록 해야 정상 실행된다. 어차피 영상분석 서비스는 Docker 나 Colab 환경에서 실행되지 않기에 해당 함수를 굳이 호출할 필요가 없다.

b. `plots.py`

70 번째 줄부터 정의된 `Annotator` 클래스는 Object Detection 결과를 화면에 표시하기 위한 클래스다. 85 번째 줄부터 정의된 `box_label` 속성에서 화면 표시 방법 및 형식을 설정할 수 있다.

c. `detect.py`

Object Detection 을 하기 위한 메인 실행코드다. 이미지 파일 뿐만 아니라 영상 파일 / 스트림도 Detection 이 가능하며, 실행에 필요한 부가 옵션들을 argument parser 로 제공하고 있다. 부가 옵션들에 대한 설명은 `parser.add_argument` 함수의 `help` 인자에서 설명하고 있다. 해당 파일로도 영상분석은 가능하나, 주차 점유 판단에 사용하기 위해서는 추가적인 알고리즘을 적용해야 하므로 이는 별도의 파일로 처리하고 있다.

d. `export.py`

모델 가중치 파일을 TensorFlow, ONNX, TensorRT 로 변환해주는 Python 코드이다. 모델 가중치 파일의 형식은 대부분 PyTorch 프레임워크 형식으로 된 pt 확장자를 사용하고 있으나, 다른 프레임워크 형식의 가중치 파일을 사용할 시 해당 코드를 실행하면 편리하다. 특히 TensorRT 프레임워크는 기존의 PyTorch 에 비해 실행속도가 빠르고 리소스 소비량이 적어 임베디드 시스템 (예. Jetson)에 사용하기 적합하다.

e. `train.py`

Yolov5 모델을 학습할 때 사용하는 Python 코드이다. 커스텀 데이터셋을 이용한 모델 학습 방법은 <https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data> 를 참고하기 바란다.

f. `requirements.txt`

Yolov5 소스코드 실행에 필요한 필수 패키지 목록이 기재된 파일이다. Python pip 가 설치되어 있으면 `pip install -r requirements.txt` 명령어로 해당 패키지들을 일괄적으로 설치할 수 있다.

3. 주차 현황 서비스용 소스코드 설명

이 항목에서는 Yolov5 모델 기반으로 주차장 내 차량을 인식하고 각 주차면 별로 차량의 점유 여부를 판단하여 주차면 DB 로 그 결과를 전송할 때 필요한 소스코드 파일에 대한 설명을 포함하고 있다. 해당 파일들은 Yolov5 소스코드의 루트 디렉토리에 포함되어 있으며, 파일 이름 및 형식은 주차장에 따라 다소 상이할 수 있으니 유의하기 바란다.

A. `config_*.yaml`

주차면 점유 판단에 필요한 각종 정보를 포함하는 설정 파일이다.

a. `src`

영상의 RTSP 스트리밍 접속 주소

b. `pos`

영상 내 점유 판단을 할 주차면의 slot id 리스트

c. `uuid`

영상 내 점유 판단을 할 주차면의 uuid 리스트

d. `roi`

주차면의 직사각형 관심 영역 (Region of Interest; ROI) 리스트, 실내 주차장에서 사용

e. `loi`

주차면의 선분 관심 영역 (Line segment of Interest; LOI) 리스트, 야외 주차장에서 사용

f. `double_rois (optional)`

이중주차가 발생하는 위치의 ROI 리스트

g. `double_slots (optional)`

이중주차로 영향을 받는 기존 주차면들의 slot id 리스트

B. `check_state.py`

주차면 점유 여부를 판단하는 알고리즘 소스 코드

a. `inout`

인식된 차량의 bounding box 와 설정한 ROI 를 이용하여 점유 여부를 판단하는 함수다. [그림 1]에서 빨간색 박스는 bounding box, 노란색 박스는 차량이 점유 중인 ROI, 초록색 박스는 차량이 비어 있는 ROI, 파란색 점선은 기준선을 나타낸다. 그러면 이 함수에서는 다음과 같은 순서로 점유 여부를 판단한다. 전체 함수는 [표 3]에서 확인 가능하다.

- i) Bounding box 가 기준선의 왼쪽에 위치한지, 오른쪽에 위치한지 판단
- ii) 기준선의 왼쪽에 위치한 bounding box 는 우측 하단점이 ROI 박스 내에 위치한지, 오른쪽에 위치한 bounding box 는 좌측 하단점이 ROI 박스 내에 위치한지 판단
- iii) 각 주차면의 점유 여부를 하나의 딕셔너리로 취합하여 반환

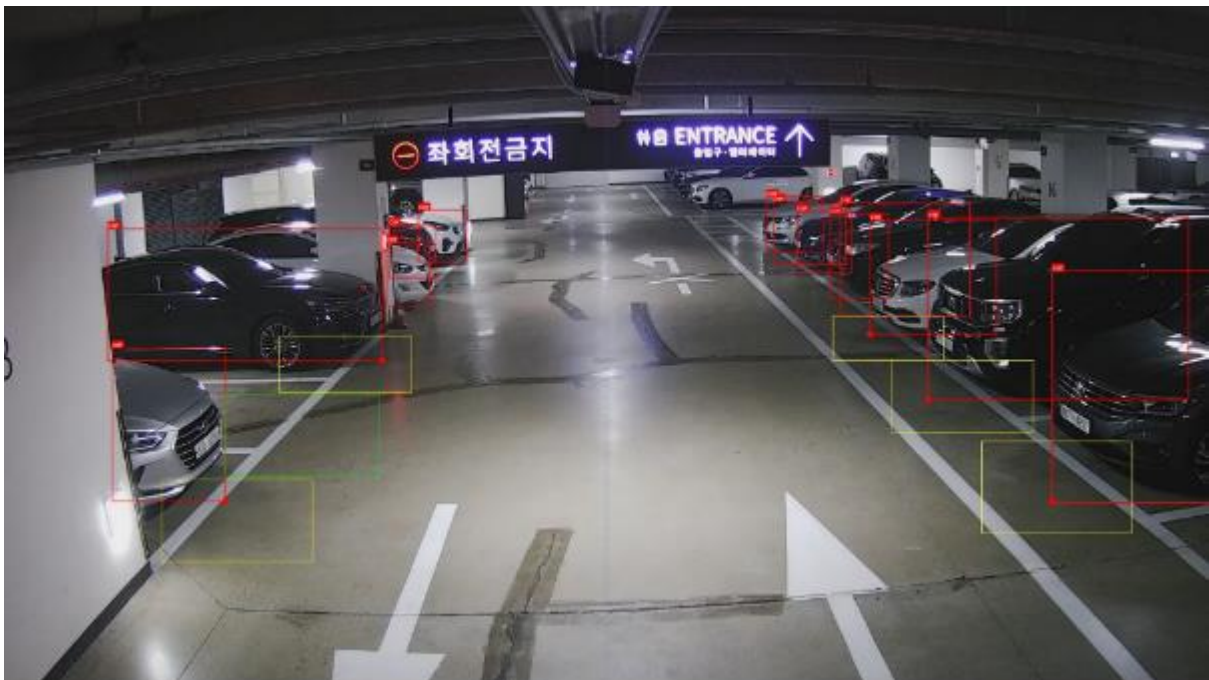


그림 1. ROI 를 이용한 주차면 점유 판단 예시

```
def inout(img, cam, bboxes, road):  
    """인식된 차량의 bounding box 와 설정한 ROI box 로 주차 점유여부 판단  
    Args:  
        img      (ndarray): 원본 이미지  
        cam      (dict): 한 카메라 설정 정보 (rtsp url, road, uuid, slot_id, roi)  
        bboxes   (List): YOLOv5 가 인식한 차량의 bounding box 좌표 리스트 (xyxy)  
        road     (int): 도로의 x 좌표 (수직선)  
  
    Returns:
```

```

dict: 한 카메라에 대한 각 주차면의 점유 상태 딕셔너리 (예: {'1-x2-y3-1': 'free', '1-
x2-y3-2': 'in'})
"""
inouts = dict() # 각 주차면별 점유 상태 딕셔너리

for idx, slot_id in enumerate(cam['pos']):
    roi = cam['roi'][idx]
    cv2.rectangle(img, (roi[0], roi[1]), (roi[2], roi[3]), GREEN, 1) # display ROI
    if not bboxes:
        # 카메라 내 인식된 차량이 한 대도 없을 때
        inouts[slot_id] = 'free'
    else:
        # 카메라 내 인식된 차량이 한 대 이상일 때
        for bbox in bboxes:
            is_left = (bbox[0] + bbox[2]) / 2 < road # 차량의 bounding box 가 도로
            # 기준으로 좌측에 있는지 확인
            is_y_inside = roi[1] < bbox[3] < roi[3] # 차량의 bounding box 의 y 좌표가
            # ROI 내부에 포함되는지 확인
            is_x_right_inside, is_x_left_inside = roi[0] < bbox[2] < roi[2], roi[0]
            < bbox[0] < roi[2]
            if is_left and is_y_inside and is_x_right_inside:
                # 차량이 도로 좌측에 주차된 경우 bounding box 의 우측 하단점 기준으로
                # 점유 판단
                cv2.rectangle(img, (roi[0], roi[1]), (roi[2], roi[3]), YELLOW, 1) #
                # 점유된 주차면의 ROI 는 노란색으로 표시
                inouts[slot_id] = 'in'
                break
            elif not is_left and is_y_inside and is_x_left_inside:
                # 차량이 도로 우측에 주차된 경우 bounding box 의 좌측 하단점 기준으로
                # 점유 판단
                cv2.rectangle(img, (roi[0], roi[1]), (roi[2], roi[3]), YELLOW, 1) #
                # 점유된 주차면의 ROI 는 노란색으로 표시
                inouts[slot_id] = 'in'
                break
            else:
                cv2.rectangle(img, (roi[0], roi[1]), (roi[2], roi[3]), GREEN, 1)
                inouts[slot_id] = 'free'

return inouts

```

표 3. **inouts** 함수 소스코드 전문

b. **occupancy_inout**

인식된 차량의 좌표와 설정한 LOI 를 이용하여 점유 여부를 판단하는 함수다. 이 함수는 주차장 천장에 카메라를 설치할 수 없는 실외 주차장에서 사용한다. **inout** 함수에서는 인식된 차량의 좌표를 직사각형의 box 형태로 나타냈지만 여기서는 차량의 좌측 상단에서 우측 하단을 잇는 선분으로 차량의 인식 여부를 표시한다. [그림 2]에서 청록색 선분은 인식된 차량을 나타내며, 빨간색 선분은 차량이 점유 중인 LOI, 초록색 선분은 차량이 비어 있는 LOI 를 나타낸다. 그러면 이 함수에서는 다음과 같은 순서로 점유 여부를 판단한다. 전체 함수는 [표 4]에서 확인 가능하다.

- i) 차량의 좌측 상단과 우측 하단을 잇는 직선과 LOI 의 직선이 교차하는지 확인

- ii) 차량의 좌측 상단 좌표를 (x_1, y_1) , 우측 하단 좌표를 (x_2, y_2) 라 하고 LOI의 양 끝점 좌표를 각각 (x_3, y_3) , (x_4, y_4) 라 할 때 다음의 수식을 만족하면 차량의 직선과 LOI가 교차한다고 판단 가능

$$(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4) \neq 0$$

- iii) 교점의 좌표를 (a, b) 라 할 때, 교점이 LOI와 차량의 직선상에 실제로 위치한지 확인하기 위해 다음 부등식을 모두 만족하면 해당 주차면에 차량이 점유된 것으로 처리

$$x_1 < a < x_2, x_3 < a < x_4, y_1 < b < y_2, y_3 < b < y_4$$

- iv) 각 주차면의 점유 여부를 하나의 딕셔너리로 취합하여 반환



그림 2. LOI를 이용한 주차면 점유 판단 예시

⚠ 주의

[그림 2]와 같이 차량 인식 (청록색) 선분을 표시하기 위해서는 2.D.b항 `plots.py`의 `Annotator.box_label` 함수에서 `cv2.rectangle` 함수 대신 `cv2.line` 함수를 사용하면 사각형 대신 선분을 그릴 수 있다. 즉, `cv2.rectangle(self.im, p1, p2, color, thickness=self.lw, lineType=cv2.LINE_AA)` 코드를 `cv2.line(self.im, p1, p2, color, thickness=self.lw, lineType=cv2.LINE_AA)`으로 교체하면 된다.

```

def occupancy_inout(annotator, bound_lines, cam):
    """객체 추적 알고리즘을 이용한 주차 점유 판단
    Args:
        annotator (:obj:`Annotator`): 영상분석 결과를 이미지에 표시하기 위한 객체
        bound_lines (list): 한 카메라에 대해 인식된 차량의 좌상단에서 우하단
        지점까지의 직선 리스트
        cam (dict): 한 카메라 설정 정보 (rtsp url, road, uuid, slot_id,
        borderline)

    Returns:
        dict: 한 카메라에 대한 각 주차면별 주차 점유 상태 딕셔너리 (예. {'1-x2-y4-1': 'in',
        '1-x2-y4-2': 'free'})
        """
    inouts = dict() # 각 주차면별 주차 점유 상태 딕셔너리

    for idx, slot_id in enumerate(cam['pos']):
        loi = cam['loi'][idx]
        cv2.line(annotator.result(), (loi[0], loi[1]), (loi[2], loi[3]), GREEN, 2,
        cv2.LINE_AA)
        if len(bound_lines) != 0:
            # 인식된 차량이 존재하는지 확인
            for bound_line in bound_lines:
                # 차량의 좌상단에서 우하단 지점까지의 직선 그리기
                x1, y1, x2, y2 = loi[0], loi[1], loi[2], loi[3]
                x3, y3, x4, y4 = bound_line[0], bound_line[1], bound_line[2],
                bound_line[3]

                if (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4) != 0:
                    # 주차선과 차량 직선이 교차하는지 확인 후 교차하면 교점의 x, y 좌표 계산
                    x = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 *
                    x4)) / ((x1 - x2) * (y3 - y4) - (x3 - x4) * (y1 - y2))
                    y = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 *
                    x4)) / ((x1 - x2) * (y3 - y4) - (x3 - x4) * (y1 - y2))
                    if (x1 <= x <= x2 or x2 <= x <= x1) and (y1 <= y <= y2 or y2 <= y <=
                    y1) and \
                    (x3 <= x <= x4 or x4 <= x <= x3) and (y3 <= y <= y4 or y4 <= y
                    <= y3):
                        # 교점이 주차선 상에 위치한지 확인
                        cv2.line(annotator.result(), (loi[0], loi[1]), (loi[2], loi[3]),
                        RED, 2, lineType=cv2.LINE_AA)
                        inouts[slot_id] = 'in'
                        # 교점이 존재하면 주차면 점유 처리, 그렇지 않으면 공면으로 처리
                        break
                else:
                    inouts[slot_id] = 'free'
            else:
                inouts[slot_id] = 'free'
        else:
            inouts[slot_id] = 'free'

    return inouts

```

표 4. occupancy_inout 함수 소스코드 전문

c. `find_intersection`

객체 추적 (Object tracking) 알고리즘을 이용하여 주차 판단 시 필요한 함수다. 객체 추적 알고리즘에서는 객체 재식별화 (Object Re-identification; Object ReID) 알고리즘으로 객체에 ID 를 부여하고, 현재와 이전 이미지를 비교하여 동일한 ID 를 갖는 객체의 이동 방향을 파악할 수 있다. [그림 3]에서 분홍색 사각형은 인식된 차량의 bounding box, 파란색 방향직선은 차량의 이동 궤적, 초록색 선분은 차량이 주차선을 통과하지 않은 주차면의 LOI, 빨간색 선분은 차량이 주차선을 통과하는 주차면의 LOI 를 나타낸다. `find_intersection` 함수는 `occupancy_inout` 함수와 동일하게 두 직선의 교차 여부 및 해당 선분 내 위치 여부를 판단하는 함수다. 차이점은 `find_intersection` 함수에서는 정지된 차량이 아닌 움직이는 차량의 궤적과 주차면 LOI 의 교차 여부를 판단한다는 점이다. [표 5]에서 이 함수 전문을 확인할 수 있다.

⚠ 참고

객체 재식별화 (Object ReID) 알고리즘의 종류는 여러 가지가 있는데, 현재 가장 많이 사용되는 알고리즘은 Mikel Broström 의 StrongSORT 알고리즘이다. 이 알고리즘은 객체 재식별화 과정에서 발생했던 객체 폐색 (occlusion) 및 다중 객체 ID 변동 (Multiple Object Tracking ID Switching; MOT ID Switching) 문제를 개선했다. StrongSORT 알고리즘의 GitHub 프로젝트는 https://github.com/mikel-brostrom/Yolov5_StrongSORT_OSNet 에서 다운받을 수 있으며, 객체 재식별화 및 추적 알고리즘에 대한 자세한 설명은 <https://gngsn.tistory.com/94> 또는 관련 문헌을 참고하기 바란다.

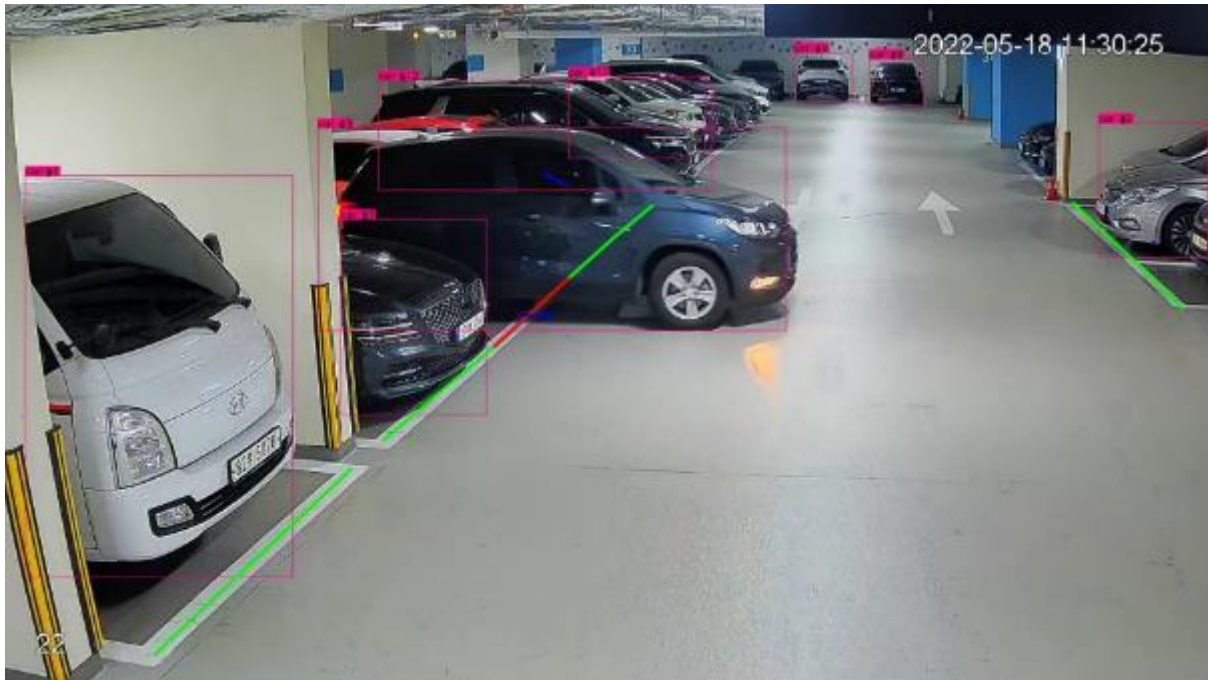


그림 3. Object tracking 알고리즘을 이용한 주차면 점유 판단 예시


```

def find_intersection(annotator, trajectories, cam, cam_seq):
    """객체 추적 알고리즘을 이용한 주차 점유 판단
    Args:
        annotator (:obj:`Annotator`): 영상분석 결과를 이미지에 표시하기 위한 객체
        trajectories (dict): 모든 카메라에 대해 이전 이미지와 현재 이미지의 차량
        bounding box 의 중심 하단 좌표
        cam (dict): 한 카메라 설정 정보 (rtsp url, road, uuid, slot_id,
        borderline)
        cam_seq (str): 카메라 ID

    Returns:
        dict: 한 카메라에 대한 각 주차면별 주차 점유 상태 딕셔너리 (예. {'1-x2-y4-1': 'in',
        '1-x2-y4-2': 'free'})
        dict: 한 카메라에 대한 각 주차면별 차량 주차선 통과 여부 딕셔너리 (True: 통과) (예.
        {'1-x2-y4-1': False, '1-x2-y4-2': True})
    """
    inouts = dict() # 각 주차면별 주차 점유 상태 딕셔너리
    inout_list = list() # 한 카메라에 대한 주차 점유 상태 리스트
    isintersects = dict() # 각 카메라별 주차선과 차량 궤적 교차 여부 딕셔너리
    for idx, slot_id in enumerate(cam['pos']):
        borderline = cam['borderline'][idx]
        cv2.line(annotator.result(), (borderline[0], borderline[1]), (borderline[2],
        borderline[3]), GREEN, 2, cv2.LINE_AA)
        if len(trajectories[cam_seq]) != 0:
            # 차량 궤적이 존재하는지 확인
            for trajectory in trajectories[cam_seq]:
                # 이전 이미지와 현재 이미지의 차량 bbox 의 중심 (하단) 좌표를 연결하는
                유향직선 그리기
                x1, y1, x2, y2 = borderline[0], borderline[1], borderline[2],
                borderline[3]
                x3, y3, x4, y4 = trajectory[0][0], trajectory[0][1], trajectory[1][0],
                trajectory[1][1]
                cv2.arrowedLine(annotator.result(), (x3, y3), (x4, y4), BLUE, 2,
                line_type=cv2.LINE_AA)

                if (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4) != 0:
                    # 주차선과 궤적이 교차하는지 확인 후 교차하면 교점의 x, y 좌표 계산
                    x = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 *
                    x4)) / ((x1 - x2) * (y3 - y4) - (x3 - x4) * (y1 - y2))
                    y = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 *
                    x4)) / ((x1 - x2) * (y3 - y4) - (x3 - x4) * (y1 - y2))
                    if (x3 <= x <= x4 or x4 <= x <= x3) and (y3 <= y <= y4 or y4 <= y <=
                    y3) and x1 <= x <= x2 and (y1 <= y <= y2 or y2 <= y <= y1):
                        # 교점이 궤적 선상에 위치한지 확인
                        isintersects[slot_id] = True
                        cv2.line(annotator.result(), (borderline[0], borderline[1]),
                        (borderline[2], borderline[3]), RED, 2, cv2.LINE_AA)
                        inouts[slot_id] = trace_inout(borderline, [x3, y3, x4, y4],
                        inouts, slot_id, cam['road'])
                        # 교점이 존재하면 차량 이동방향으로 점유 여부 판단, 그렇지 않으면
                        None 으로 처리 (판단불가)
                        break
                    else:
                        isintersects[slot_id] = False
                        inouts[slot_id] = None
                else:

```

```

        isintersects[slot_id] = False
        inouts[slot_id] = None
    else:
        inouts[slot_id] = None
        inout_list = list(inouts.values())
    return inouts, isintersects

```

표 5. `find_intersection` 함수 소스코드 전문

d. `trace_inout`

`find_intersection` 함수로 차량의 이동 궤적과 주차면 LOI 와의 교차 여부를 판단한 후, 차량의 이동 방향을 파악하여 주차면 내 차량의 진출입 여부를 판단하는 함수이다. [그림 3]에서 주차장의 중앙 통로를 기준으로 차량이 좌측에 위치한지 혹은 우측에 위치한지 확인한 후, 차량 궤적의 x 좌표의 이동 방향을 계산하여 통로의 좌측에 위치한 차량은 궤적 방향이 오른쪽을 향하면 출차, 왼쪽을 향하면 입차로 판단한다. 반대로 통로의 우측에 위치한 차량은 궤적 방향이 왼쪽을 향하면 출차, 오른쪽을 향하면 입차로 판단한다. 이 함수는 차량의 이동 궤적과 주차면 LOI 가 실제로 교차하였을 때, 즉 차량이 주차선을 통과하고 있을 때만 작동하기 때문에 단독으로 구현이 불가능하며, 반드시 `find_intersection` 함수를 먼저 구현해야 한다. [표 6]에서 [그림 3]의 상황일 때 이 함수를 실행했을 때 결과를, [표 8]에서 함수 전문을 확인할 수 있다.

```

Parking state: {'1-x2-y4-1': None, '1-x2-y4-2': None, '1-x2-y4-3': None,
'1-x2-y5-1': None, '1-x2-y5-2': None,
'1-x3-y5-3': 'free', '1-x3-y5-2': None, '1-x3-y5-1': None}
Intersection: {'1-x2-y4-1': False, '1-x2-y4-2': False, '1-x2-y4-3':
False, '1-x2-y5-1': False, '1-x2-y5-2': False,
'1-x3-y5-3': True, '1-x3-y5-2': False, '1-x3-y5-1': False}

```

표 6. [그림 3] 상황에서 `trace_inout` 함수 구현 결과

e. `double_inout`

이중주차로 인하여 기존의 주차면에 주차된 차량이 인식되지 못하는 문제를 해결하는 함수이다. [그림 4]에서와 같이 이중주차 발생 위치 ROI (보라색 박스)를 설정하여 해당 위치에 차량이 주차했다고 인식하면 이중주차로 영향을 받는 기존 주차면은 YoloV5 모델의 인식 결과와 관계없이 무조건 점유된 것으로 처리한다. [표 7]에서 함수 전문을 확인할 수 있다.

```

def double_inout(img, cam, bboxes, road):
    """인식된 차량의 bounding box 와 설정한 ROI box 로 이중주차 여부 판단
    Args:
        img      (ndarray): 원본 이미지
        cam      (dict): 한 카메라 설정 정보 (rtsp url, road, uuid, slot_id, roi)
        bboxes   (list): YoloV5 가 인식한 차량의 bounding box 좌표 리스트 (xyxy)
        road     (int): 도로의 x 좌표 (수직선)

    Returns:
        dict: 한 카메라에 대한 각 주차면의 점유 상태 딕셔너리 (예: {'1-x2-y3-1': 'free', '1-x2-y3-2': 'in'})
    """
    for idx, double_roi in enumerate(cam['double_rois']):
        if not bboxes:
            # 카메라 내 인식된 차량이 한 대도 없을 때
            break
        else:
            # 카메라 내 인식된 차량이 한 대 이상일 때
            for bbox in bboxes:
                is_left = (bbox[0] + bbox[2]) / 2 < road # 차량의 bounding box 가 도로
                # 기준으로 좌측에 있는지 확인
                is_y_inside = double_roi[1] < bbox[3] < double_roi[3] # 차량의 bounding box 의
                # y 좌표가 이중주차의 ROI 내부에 포함되는지 확인
                is_x_right_inside, is_x_left_inside = double_roi[0] < bbox[2] < double_roi[2],
                double_roi[0] < bbox[0] < double_roi[2]
                if is_left and is_y_inside and is_x_right_inside:
                    # 차량이 도로 좌측에 주차된 경우 bounding box 의 우측 하단점 기준으로 점유
                    # 판단
                    cv2.rectangle(img, (double_roi[0], double_roi[1]), (double_roi[2],
                    double_roi[3]), PURPLE, 1) # 이중주차 발생시 해당 구역의 ROI 는 보라색으로 표시
                    for double_slot in cam['double_slots'][idx]: # 이중주차로 영향을 받는 기존의
                    # 주차면은 모두 점유된 것으로 강제 처리
                    cam['state'][double_slot] = 'in'
                    break
                elif not is_left and is_y_inside and is_x_left_inside:
                    # 차량이 도로 우측에 주차된 경우 bounding box 의 좌측 하단점 기준으로 점유
                    # 판단
                    cv2.rectangle(img, (double_roi[0], double_roi[1]), (double_roi[2],
                    double_roi[3]), PURPLE, 1) # 이중주차 발생시 해당 구역의 ROI 는 보라색으로 표시
                    for double_slot in cam['double_slots'][idx]: # 이중주차로 영향을 받는 기존의
                    # 주차면은 모두 점유된 것으로 강제 처리
                    cam['state'][double_slot] = 'in'
                    break
                else:
                    continue
    return cam['state']

```

표 7. double_inout 함수 소스코드 전문



그림 4. `double_inout` 함수 적용 결과 예시

```
def trace_inout(borderline, trace, inouts, slot_id, road):
    """객체 추적 알고리즘을 이용한 주차 점유 판단
    Args:
        borderline (list): 한 주차선의 양 끝점 (x1, y1, x2, y2)
        trace      (list): 주차선과 교차하는 차량 궤적의 양 끝점 (x1, y1, x2, y2)
        inouts     (dict): 한 카메라에 대한 각 주차면별 주차 점유 상태 딕셔너리 (예. {'1-x2-y4-1': 'in', '1-x2-y4-2': 'free'})
        slot_id    (str): 주차면 ID (예. '1-y2-x3-2')
        road       (int): 도로의 x 좌표 (수직선)

    Returns:
        int: 한 주차면의 점유 상태 (1: 점유, 0: 공면)
    """
    if (borderline[0] + borderline[2]) / 2 < road:
        # 주차면이 기준선 좌측에 위치한 경우
        if trace[0] - trace[2] > 0 or trace[1] - trace[3] > 0:
            # 차량이 왼쪽 또는 위쪽으로 이동한 경우
            inouts[slot_id] = 'in'
        else:
            inouts[slot_id] = 'free'
    else:
        # 주차면이 기준선 우측에 위치한 경우
        if trace[0] - trace[2] < 0 or trace[1] - trace[3] > 0:
            inouts[slot_id] = 'in'
        else:
            inouts[slot_id] = 'free'
    return inouts[slot_id]
```

표 8. `trace_inout` 함수 소스코드 전문

C. `dataloader.py`

카메라 영상을 읽고 그 영상을 YoloV5 모델에 전달하는 소스코드로, `LoadData` 클래스로 구현되어 있다. `config*.yaml` 파일의 `src` 항목에 기록된 모든 카메라의 RTSP 스트리밍 접속 주소를 가져와 OpenCV를 이용하여 영상 프레임을 읽는다. 이 과정은 비동기 스레드로 구현되어 영상 분석 중에도 다음 영상을 순차적으로 계속 로딩할 수 있다.

D. `detector.py`

영상분석을 위한 YoloV5 모델의 매개변수 값을 설정하고, 이미지 전처리 (preprocessing) 후 분석 결과를 bounding box 또는 선분으로 이미지 상에 표시하는 소스코드로, `Detector` 클래스로 구현되어 있다.

a. `_make_img`

이미지는 여러 개의 픽셀이라는 작은 점으로 구성되어 있으며, 하나의 픽셀은 빨강(R), 파랑(B), 초록(G)의 3 가지 색상 값을 혼합하여 하나의 색상을 표현한다. 픽셀의 색상 값은 이 3 가지 값을 벡터로 결합하여 나타내며 각 값은 0에서 255 사이의 값을 갖는데, OpenCV에서는 (파랑, 초록, 빨강), 즉 (B,G,R)의 순서로 나타낸다. 이 함수는 BGR 순서로 표현된 픽셀 값을 RGB 순서로 변환하고, 그 값을 0 과 1 사이의 값으로 표준화하는 전처리 과정을 수행한다.

b. `detect`

설정한 YoloV5 모델로 인식한 결과를 이미지 상에 표현하는 함수다. 인식 결과를 최대한 정확하게 표현하기 위해 최대 억제 (non-max suppression; NMS)를 수행하며 이를 수행하기 위해 필요한 Intersection of Union (IoU) 값과 인식 결과의 confidence 값을 적절히 조정할 수 있다.

! 참고

YoloV5는 특정 물체 주변에 여러 개의 boundary 좌표 (물체의 좌측 상단과 우측 하단 좌표) 값을 산출하는데, non-max suppression (NMS)는 여러 개의 boundary 좌표 중 가장 confidence 가 높은 값을 우선 선택하여 이미지 상에 표현한다. 영상분석 시에는 이 confidence 의 하한 (lower-bound)값을 설정할 수 있는데, 이 값보다 낮은 confidence 를 갖는 값은 인식 결과에서 제거한다. 또한 IoU의 상한 (upper-bound)값도 설정할 수 있는데, 하나의 bounding box 좌표값을 기준으로 다른 bounding box 좌표값과의 IoU를 계산하여 설정 값보다 높은 bounding box를 제거한다. 즉, IoU 값이 높을수록 객체 주변의 boundary 좌표 개수는 줄어들며, confidence 값이 낮을수록 boundary 좌표 개수는 많아지게 된다. 그러나 confidence 값을 지나치게 낮게 설정하면 원하지 않는 객체도 인식하게 되어 인식 정확도가 저하될 수 있으므로 여러 번의 테스트를 통해 최적의 값을 설정하는

과정이 필요하다. NMS 에 관한 상세 설명은 <https://ctkim.tistory.com/98> 또는 관련 문헌 등을 참고하기 바란다.

E. `run_final.py`

영상분석 서비스를 실행하는 메인 소스코드다. `parse_opt` 함수에서 실행에 필요한 매개변수의 값을 지정할 수 있다. 매개변수의 종류는 `detect.py` 의 `parse_opt` 함수와 동일하다. 이 매개변수들은 `run` 함수로 전달되며, `run` 함수에서는 `Detector` 클래스와 `LoadData` 클래스에서 객체를 각각 생성한 후, 영상분석을 수행하여 `check_state.py` 의 주차면 점유 판단 함수로 주차면 점유 결과를 산출한다. 산출된 결과는 `transmit_server.py` 의 `sendstate2server` 함수를 이용하여 주차면 API 서버로 전송하여 위치마일 앱과 주차 현황이 연동되도록 한다. 다음 하위 항목에서 `run_final.py` 실행 시 호출하는 함수들에 대한 설명을 담고 있다.

a. `main`

`run` 함수를 실행하는 `main` 함수이다. `run` 함수를 실행하기 전 `parse_opt` 함수를 호출하여 실행 매개변수 옵션을 로딩한다. [표 9]에서 `main` 함수의 전문을 확인할 수 있다. 영상 분석할 카메라가 매우 많으면 멀티프로세싱을 이용하여 `run` 함수를 여러 개 실행할 수 있다. 멀티프로세싱은 `concurrent.futures` 패키지 내 `ProcessPoolExecutor` 클래스를 import 한 후, [표 9]에서 정의된 `main` 함수 내에서 `run` 함수를 주석 처리하고 `ProcessPoolExecutor` 가 포함된 `with` 구문을 주석 해제하여 활성화하면 여러 개의 `run` 함수를 실행할 수 있다.

⚠ 주의

Python에서는 멀티프로세싱 구현을 위해 `multiprocessing.Pool` 클래스 또는 `concurrent.futures` 패키지 내 `ProcessPoolExecutor` 클래스를 사용할 수 있다. 두 클래스 모두 멀티프로세싱 구현이 가능하지만 각 프로세스의 중단 또는 예외 처리 부분에서 `ProcessPoolExecutor` 클래스가 좀 더 유리하다. 특히 영상분석 프로세스는 CPU와 RAM 뿐만 아니라 GPU 리소스도 사용하기 때문에 효율적인 GPU 리소스 관리를 위해 `ProcessPoolExecutor` 클래스를 사용하고 있다. `multiprocessing.Pool` 클래스와 `ProcessPoolExecutor` 클래스의 차이점에 대한 자세한 설명은 <https://superfastpython.com/multiprocessing-pool-vs-processpoolexecutor/#Comparison of Pool vs ProcessPoolExecutor>를 참고하기 바란다. 멀티프로세싱 구현 시에는 싱글프로세싱보다 더 많은 시스템 리소스를 사용하기 때문에 리소스 부족으로 인해 프로세스가 종료되지 않도록 적절한 개수의 프로세스를 사용해야 한다.

```
def main(opt):
    run(1, 'config_Ansan_water_top.yaml', True, 'occupancy', **vars(opt))
    # with ProcessPoolExecutor() as executor:
    #     procs = [executor.submit(run, 1, 'config_Anyang2_SKV1.yaml', 'occupancy',
    **vars(opt)),
    #               executor.submit(run, 1, 'config_Anyang2_SKV1.yaml', 'occupancy',
    **vars(opt))]

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

표 9. main 함수 소스코드 전문

b. `insert_data` (optional)

영상분석 결과를 데이터 분석 서버로 전송하기 위해 결과 정보를 딕셔너리 형태로 취합하는 함수이다. 전송할 결과 항목에 대한 설명은 [표 10]에 기재하였다.

키 값	설명	예시
cctv-id	카메라 ID	'CCTV03'
model-name	영상분석의 모델명	'yolov5'
model-version	모델의 API 버전	6.0
pks-id-list	주차면 ID 리스트	['1-x2-y3-2', '1-x2-y3-1']
roi-box-coord-list	주차면에 대응하는 ROI 좌표 리스트	[[320, 590, 367, 630], [231, 583, 278, 774]]
predicted-by-model-list	모델이 분석한 주차면 점유 결과 리스트	{ '1-x2-y3-2': 'free', '1-x2-y3-1': 'in' }
bounding-box-coord-list	모델이 분석한 차량의 bounding box 좌표 리스트	[[503, 298, 769, 532], [310, 487, 353, 690]]
img-path	영상분석 결과 이미지 저장 경로	'/home/ves/anyang2-parking-inout/runs/detect/exp/CCTV03.jpg'
region	주차장 소재지	'Anyang'
pk-name	주차장 이름	'Anyang2_SKV1'

표 10. 데이터 서버 전송용 데이터 항목 및 설명

F. `transmit_server.py`

영상분석으로 얻은 주차면 점유 결과를 주차면 현황 API 서버로 전송하는 소스코드다. 전송 방식은 HTTP Requests 의 메소드를 사용하여 점유 결과를 JSON 형식으로 변환한 후 전송한다. 전송에 사용되는 함수에 대한 상세 설명을 하위 항목에서 설명한다.

a. `sendstate2server`

주차면 점유 상태를 주차면 현황 API 서버로 전송하는 함수다. 주차면 현황 API 서버로 전송할 때는 API 서버 주소 뒤에 각 주차면의 uuid 를 추가하여 각 주차면 별

데이터를 호출한 후, 주차면 현황을 알려주는 ‘slot_status’ 키 값에 주차면 점유 현황 값 (‘in’, ‘free’)을 전송한다. 주차면 점유 현황 값 전송은 각 주차면 별로 개별적으로 전송해야 하므로 네트워크 리소스를 많이 사용하게 되는데, 이는 영상분석 프로세스의 수행 시간에 많은 영향을 주기 때문에 최적화가 필요하다. 따라서 최초로 영상분석 프로세스를 실행할 때는 주차면 현황 API 서버에서 기존의 주차면 현황 데이터를 수신하여 현재 분석한 점유 현황 값과 상이할 때만 업데이트를 수행하고, 이후에는 이전의 영상분석 결과와 비교하여 그 결과가 상이할 때만 업데이트를 수행하여 네트워크 사용량을 줄였다. 전체 소스코드는 [표 11]에서 확인 가능하다.

```
def sendstate2server(cam, site):
    """주차 점유 상태 전송
    주차면의 점유 상태를 앱 서비스용 API 서버에 전송
    Args:
        cam      (dict): 한 카메라에 대한 정보 (기준 도로 좌표, 영상 접속 주소, 주차면별 uuid,
        slot_id, ROI 좌표)
        site      (str): 주차장 이름 (skv1-ay2: 안양2 차 SKV1, hobanpark: 호반파크 2 관,
        cheonho: 천호/강동역 공영주차장, ansan: 안산 고잔동 공영주차장, 기타 등)

    """
    headers = {'Content-Type': 'application/json'}
    url = f'{"http" if site == "cheonho" or site == "ansan" else "https"}://{site}.watchmile.com/api/v1/parking/slot/'
    try:
        for idx, state in enumerate(cam['state'].values()):
            uuid = cam['uuid'][idx]
            if 'prev_state' in cam:
                # 최초 프로세스 실행 시를 제외하고 이전 영상 프레임에서의 분석 결과 호출
                prev_state = cam['prev_state'][cam['pos'][idx]]
            else:
                # 최초 프로세스 실행 시 이전의 주차 상태를 API 서버에서 호출
                slot_data = get(url + uuid)
                prev_state = slot_data.json()["result"]["slot_status"]
            if state != prev_state:
                put(url + uuid, {'slot_status': state}, headers)
                LOGGER.info(f'Parking state updated in {cam["pos"][idx]}')
                continue
            if state != prev_state:
                # 현재 주차 점유 상태와 이전의 상태가 상이할 때만 API 서버로 결과 전송
                put(url + uuid, {'slot_status': state}, headers)
                LOGGER.info(f'Parking state updated in {cam["pos"][idx]}')
            else:
                continue
    except Exception as e:
        LOGGER.exception(e)
    pass
```

표 11. sendstate2server 함수 소스코드 전문

b. put, get

Python은 `request` 패키지에서 HTTP API 메소드를 제공하고 있다. 물론 이 메소드를 그대로 사용해도 되지만 예외 처리의 용이성을 위해 별도의 함수로 정의하여 사용한다. 전송이 성공하면 “`transmission success!`”라는 로그 메시지를 남기도록 하여 로그 파일에서 이를 확인할 수 있도록 하였다. 또한 오류 발생 시 에러 메시지도 남겨 오류 발생 원인을 파악할 수 있도록 하였다. 전체 소스코드는 [표 12]에서 확인할 수 있다.

```
# 관제 서버에 주차 상태를 전송
def put(url1, data, headers):
    try:
        r = requests.put(url1, data=json.dumps(data), headers=headers)
        if r.status_code != 200: # 전송 성공시 상태 코드 출력
            LOGGER.info("HTTP Error code: ", r.status_code)
        else:
            LOGGER.info("transmission success!")
    # 오류 발생시 예외처리
    except requests.exceptions.Timeout as errd:
        LOGGER.exception("Timeout Error : ", errd)
    except requests.exceptions.ConnectionError as errc:
        LOGGER.exception("Error Connecting : ", errc)
    except requests.exceptions.HTTPError as errb:
        LOGGER.exception("Http Error : ", errb)
    # Any Error except upper exception
    except requests.exceptions.RequestException as erra:
        LOGGER.exception("AnyException : ", erra)

# 주차면 API 서버에서 기존 주차면 상태 수신
def get(url1):
    try:
        r = requests.get(url1)
        if r.status_code != 200: # 전송 실패시 상태 코드 출력
            LOGGER.info("HTTP Error code: ", r.status_code)
        return r
    # 오류 발생시 예외처리
    except requests.exceptions.Timeout as errd:
        LOGGER.exception("Timeout Error : ", errd)
    except requests.exceptions.ConnectionError as errc:
        LOGGER.exception("Error Connecting : ", errc)
    except requests.exceptions.HTTPError as errb:
        LOGGER.exception("Http Error : ", errb)
    # Any Error except upper exception
    except requests.exceptions.RequestException as erra:
        LOGGER.exception("AnyException : ", erra)
```

표 12. `put`, `get` 함수 소스코드 전문

G. `verify.py`

영상분석 모델의 정확도는 100%가 될 수 없다. 따라서 여러 번 영상분석을 수행하면 동일한 이미지 또는 동일한 주차 상황임에도 불구하고 일부 차량을 인식할 수 없는 경우가 간혹 발생할 수 있다. 이와 같은 상황이라면 주차면 현황 페이지 또는 위치마일 앱의 주차면 현황 이미지에서 주차면의 점유 상황이 잠시 변화하게 된다.

이러한 상황에서는 주차 안내 중 주차를 하지 않음에도 불구하고 주차가 완료되거나 안내 경로가 불필요하게 변경되는 오류가 발생할 수 있다. 따라서 이와 같은 오류를 해결하기 위해 동일한 카메라에 대해 여러 번 영상분석을 반복한 후, 각 주차면 별로 점유 현황을 취합하여 최종 주차 상태를 다수결로 결정하는 알고리즘을 개발했다. 예를 들어, 영상분석을 5 회 반복하였을 때 ‘1-x2-y3-1’ 주차면의 점유 상태가 각각 ‘in’, ‘in’, ‘free’, ‘in’, ‘free’로 집계되었다면 최종 주차 상태는 ‘in’으로 판단하여 이 결과를 API 서버로 전송한다. 이 알고리즘을 `verify` 라는 함수로 구현하였으며 전체 소스코드는 [표 13]에서 확인할 수 있다.

```
def verify(cam, cam_seq, state, num=5):
    if num % 2 == 0: # 검증 결과가 동물이 나오는 것을 방지하기 위해 검증 횟수를 홀수 번으로
        변경
        num -= 1

    state[cam_seq].append(list(cam['state'].values())) # 각 카메라 별 주차 상태 리스트
    요소 추가
    if len(state[cam_seq]) > num:
        state[cam_seq].popleft()
    # LOGGER.info(f'{cam_seq} - Verifying parking state : {state[cam_seq]}')

    if len(state[cam_seq]) == num: # 동일 카메라에 대해 5 번 영상분석을 수행했을 때
        verify_state = [list() * len(cam['state']) for x in range(len(cam['state']))]
        for i in range(len(state[cam_seq][0])): # 각 주차면 별 주차 상태를 리스트로 저장
            for j in range(num):
                verify_state[i].append(state[cam_seq][j][i])

        final_state = dict() # 최종 주차 상태를 저장하는 리스트 초기화
        for i, states in enumerate(verify_state): # 각 주차면 별 주차 상태 리스트에서 최종
            주차 상태를 다수결로 결정
            if verify_state[i].count('free') > verify_state[i].count('in'):
                final_state[cam['pos'][i]] = 'free'
            else:
                final_state[cam['pos'][i]] = 'in'

        LOGGER.info(f'{cam_seq} - Final Parking State: {final_state}')
        return final_state

    else:
        return None
```

표 13. `verify` 함수 소스코드 전문

4. 영상분석 서비스 실행 방법

이 항목에서는 영상분석 서비스를 실행하기 위한 방법을 설명한다.

A. 서버 OS 설치 및 환경설정

영상분석 서비스의 실행은 기본적으로 리눅스 OS 환경에 최적화되어 있다. 윈도우나 Mac 에서도 실행은 가능하나, 일부 부가 패키지는 리눅스에서만 제공하고 있어 리눅스에서 실행하는 것을 권고한다. 서버 내 리눅스를 설치하는 방법은 [부록 1]에 상세히 설명되어 있다.

B. config_*.yaml 파일 작성

영상분석에 필요한 기본 정보를 yaml 형식으로 입력하고 config_*.yaml 파일로 저장한다. 여기서 *은 주차장 이름이다. [표 14]에서 config_*.yaml 파일의 작성 예시를 확인할 수 있다.

```
CCTV03:
  pos:
    - 1-x7-y6-3
    - 1-x7-y6-2
    - 1-x7-y6-1
    - 1-x7-y5-2
    - 1-x7-y5-1
  road: 1150
  roi:
    - [0, 750, 400, 1034]
    - [230, 520, 630, 750]
    - [460, 370, 770, 520]
    - [660, 263, 920, 349]
    - [740, 200, 970, 263]
  src: rtsp://admin:123456@172.10.30.2/unicast/c16/s0/live
  uuid:
    - a79c8efe-0a2b-11ec-9c5c-4616e5ba6420
    - a7999113-0a2b-11ec-9c5c-4616e5ba6420
    - a796a6c4-0a2b-11ec-9c5c-4616e5ba6420
    - a793a2e6-0a2b-11ec-9c5c-4616e5ba6420
    - a790be1d-0a2b-11ec-9c5c-4616e5ba6420
```

표 14. config_*.yaml 파일 내용 일부

a. ROI / LOI 설정

ROI 또는 LOI 값은 이미지에서 해당 주차면에 대응하는 영역의 좌측 상단 좌표와 우측 하단 좌표 값을 하나의 리스트로 입력한다. [그림 5]에서처럼 이미지의 좌표 값은 알씨 또는 윈도우 그림판 프로그램 등으로 확인 가능하다. 예를 들어, 좌측 상단 좌표를 (a_1, b_1) , 우측 하단 좌표를 (a_2, b_2) 라 하면 ROI 또는 LOI 값은 $[a_1, b_1, a_2, b_2]$ 라는 리스트가 된다. 이를 반복하여 하나의 카메라에 설정할 ROI 또는 LOI 값을 config_*.yaml 파일 내 roi 또는 loi 항목에 추가한다.



그림 5. 알씨를 이용한 이미지의 픽셀 좌표값 확인

b. slot id, uuid 입력

ROI 또는 LOI 설정이 완료되면 설정한 주차면에 대응하는 slot id와 uuid를 찾아 `config*.yaml` 파일의 `pos`와 `uuid` 항목에 각각 추가한다.

c. 카메라의 RTSP 스트리밍 주소 입력

각 카메라에 대응하는 RTSP 스트리밍 주소를 `config*.yaml` 파일의 `src` 항목에 추가한다.

d. road 값 입력 (optional)

`inouts` 또는 `trace_inout` 함수를 이용하여 주차 판단 시 필요한 값이다. 이미지에서 주차장 통로의 x 좌표 값을 `config*.yaml` 파일의 `road` 항목에 추가한다.

e. double_roi (optional)

이중주차 처리 알고리즘 사용 시 이중주차가 발생하는 위치에 차량이 주차하는지 판단하기 위해 필요한 값이다. 값 설정 방법은 기존의 ROI 설정 방법과 동일하다. 이 값들을 `double_roi` 항목에 추가한다.

f. double_pos (optional)

이중주차 처리 알고리즘 사용 시 이중주차 발생 시 영향을 받는 기존 주차면의 `slot_id` 목록 리스트이다. 이중주차가 발생하는 주차면 당 `slot_id`를 하나의 리스트로 작성하여 `double_pos` 항목에 추가한다.

C. `run_final.py` 실행 매개변수 설정

3.E 절에서 설명했듯이 `run_final.py` 파일에는 실행 매개변수가 있다. 영상분석 서비스를 위해 대부분 기본값을 사용하면 되나 일부 매개변수는 특정 값을 설정해야 한다. 이 항목에서는 설정이 필요한 매개변수와 그 설정 값에 대해 설명한다.

! 참고

Argument parser 를 이용하여 매개변수 값을 설정할 때 매개변수가 두 개의 대시 기호 뒤에 하나의 대시가 있으면 대시 앞의 구문만 입력해도 된다. 예를 들어, ‘--conf-thres’의 매개변수 값을 0.25 라고 설정하고 싶다면 원래는 ‘--conf-thres 0.25’라고 입력해야 되나 간략하게 ‘--conf 0.25’라고 입력해도 된다. 단, ‘--conf-thres’ 매개변수 이외에 ‘--conf-x’라는 매개변수, 즉, ‘--conf’로 시작하는 다른 매개변수가 정의되어 있을 때 간략하게 쓰면 오류가 발생한다. 설정한 매개변수가 ‘--conf-thres’인지 ‘--conf-x’인지 구별할 수 없기 때문이다.

Argument parser 에 대한 자세한 설명은 <https://engineer-mole.tistory.com/213> 를 참고하기 바란다.

a. --weights

PyTorch 프레임워크 모델 가중치 파일의 경로를 설정하는 매개변수다. 차량 인식을 위해서는 YoloV5의 기본 모델 대신 커스텀 데이터셋을 통해 학습한 모델을 사용하기 때문에 해당 모델 가중치 파일이 저장된 경로를 입력한다.

b. --conf-thres

NMS 과정에 필요한 인식 결과의 confidence 하한 값을 설정하는 매개변수다. 기본값은 0.25 로 설정되어 있는데, 0.15 까지 낮춰 설정하는 것이 좋다. 여러 차량이 밀집하여 주차되어 있을 때 각 차량들을 잘 인식할 수 있다.

c. --iou-thres

NMS 과정에 필요한 인식 결과의 IoU 상한 값을 설정하는 매개변수다. 기본값은 0.45 로 설정되어 있는데, 0.5 로 높여 설정하는 것이 좋다. confidence 값 설정과 동일한 이유다.

d. --device

GPU 사용 시 설정하는 매개변수다. 값은 숫자로 설정하는데, 0 번부터 시작한다. 즉, GPU 가 하나만 있으면 설정값은 0 이고 여러 개의 GPU 를 사용하고 싶으면 0 2 등의 값으로 설정할 수 있다.

e. --exist-ok

영상분석 결과를 저장할 때 실행 시마다 별도의 디렉토리를 생성하지 않도록 설정하는 매개변수다. 기본적으로 영상분석 결과는 프로젝트 내 `runs/detect` 라는 디렉토리 내 실행 시마다 하위 디렉토리로 `exp{n}`이라는 디렉토리를 생성하여 저장된다. 여기서

{n}은 실행에 따라 순차적으로 증가하는 숫자다. 여기서는 영상분석 서버 내의 저장 공간 관리를 위해 이 매개변수를 설정하여 `exp` 외에 다른 디렉토리를 생성하지 않도록 설정한다.

f. `--line-thickness`

Yolov5 로 인식한 결과를 이미지에 표시할 때 bounding box 또는 선분의 굵기를 지정하는 매개변수다. 기본값은 3 포인트로 지정되어 있는데, 좀 두껍게 보이기 때문에 1 로 설정해도 상관없다.

g. `--send-mqtt`

영상분석 결과를 MQTT 를 통해 influxDB 로 전송할 때 사용하는 매개변수다. 해당 기능에 대한 자세한 설명은 5 절을 참고하기 바란다.

h. `--service`

영상분석 결과를 주차면 API 서버에 전송할 때 사용하는 매개변수다. 해당 매개변수를 사용하지 않으면 테스트 용도로만 사용할 수 있다.

i. `--config-file`

`config_*.yaml` 파일의 경로를 지정하는 매개변수다. 기본적으로 각 주차장에 맞는 yaml 파일 경로가 기본값으로 지정되어 있으나, 테스트 용도로 다른 yaml 파일을 로딩하고자 할 때 해당 매개변수를 사용한다.

j. `--mode (optional)`

멀티프로세싱으로 여러 개의 영상분석 프로세스를 가동할 때 특정 카메라 정보 그룹의 데이터를 추출하기 위한 매개변수다. 예를 들어, 총 200 대의 카메라 중 20 대의 카메라의 정보만 로딩하고 싶을 때 해당 매개변수를 사용하면 된다. 물론 `run` 함수 내부에 특정 카메라 정보만 로딩할 수 있도록 실행 전 조건을 설정해야 한다.

D. 작업 스케줄러를 이용한 자동 실행 스크립트 작성

작업 스케줄러를 이용하면 영상분석을 수행하는 `run_final.py` 파일을 자동으로 실행하고 종료할 수 있다. 또한 영상분석 수행 결과를 로그 파일로 저장하여 프로세스에 문제가 발생했을 때 파일을 열어 쉽게 확인할 수 있다.

a. `runCheck.sh`

자동으로 영상분석 프로세스를 실행하고 종료할 수 있는 리눅스용 쉘 스크립트 파일이다. 이 스크립트의 수행 작업은 다음과 같은 순서로 진행한다. 스크립트 파일 전문은 [표 15]에서 확인할 수 있다.

- i) 컴퓨터가 최초로 부팅된 시점에서 얼마나 시간이 흘렀는지 확인하여 3 분 이상 지나면 다음 작업을 수행한다.

- ii) 시스템 환경 변수를 호출한다. Anaconda 가상환경을 사용하면 Anaconda 실행 파일도 같이 호출한다.
- iii) 스크립트 파일 경로, 현재시각 (YYYY-mm-dd HH:MM:SS 형식), 로그 파일의 저장 경로를 설정한다.
- iv) 매 n 시간 정각마다 영상분석 프로세스를 종료한 후 다음 작업을 수행한다.
- v) 영상분석 프로세스가 실행 중인지 확인하여 그렇지 않으면 실행한다. 이 때 실행결과를 로그 파일에 기록한다.

```
#!/bin/bash
minutes=`awk '{print $0/60;}' /proc/uptime`;
if [ $(echo "3 > $minutes" | bc) -ne 0 ]; then
    echo "$minutes";
    exit 0;
fi

export
PATH=:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games:/usr/local/games:/snap/bin
source ~/anaconda3/etc/profile.d/conda.sh

SCRIPT=`realpath $0`
SCRIPT_PATH=`dirname $SCRIPT`
# 또는
#SCRIPT_PATH=$( cd "$(dirname "$0")" ; pwd )
DATE=$(date "+%Y-%m-%d_%H:%M:%S")
LOG_PATH="$SCRIPT_PATH/shellLog"

nowHour=$((10#`date +%H`%6));
nowMinutes=$((10#`date +%M`%60));
#nowMinutes=$((date +%M) | bc);
if [ $nowMinutes -eq 0 ] && [ $nowHour -eq 0 ]; then
    pkill -9 -ef run_final
    echo "$DATE run hour Kill Success!" >> "$LOG_PATH"/run_restart.log
    sleep 1;
fi

PYTHON_CHECK_1=`ps -ef | grep -v "grep" | grep "python3 ./run_final" |
wc -l`
if [ "$PYTHON_CHECK_1" -lt 1 ]; then
    conda activate inout && cd ~/ansan-parking-inout
    mkdir -p "$LOG_PATH"/debug/$(date "+%Y-%m-%d")
    nohup python3 ./run_final.py --weights yolov5_topview_v1.pt --
conf-thres 0.2 --iou 0.45 --device 0 --nosave --exist --line 2 --service
> "$LOG_PATH"/debug/$(date "+%Y-%m-%d")/run_debug-$(date "+%H%M").log
2>&1 &
    sleep 5;
    PYTHON_CHECK_1=`ps -ef | grep -v "grep" | grep
"python3 ./run_final" | wc -l`
    if [ "$PYTHON_CHECK_1" -lt 1 ]; then
```

```

        echo "$DATE run Process Restart Failed!" >>
"$LOG_PATH"/run_restart.log
    else
        echo "$DATE run Process Restart Success!" >>
"$LOG_PATH"/run_restart.log
    fi
else
    echo "$DATE run Process Alive" >> /"$LOG_PATH"/run_alive.log
fi

```

표 15. 영상분석 프로세스 자동 실행 쉘 스크립트 파일 전문 (`runCheck.sh`)

b. 작업 스케줄러 등록

`runCheck.sh` 파일을 저장하면 이 파일을 리눅스의 작업 스케줄러 (crontab)에 등록한다. 터미널에 ‘`crontab -e`’ 명령어를 입력하면 [그림 6]와 같이 텍스트 편집 창으로 전환되며 이 파일의 맨 아랫줄에 ‘`* * * * * bash ~/runCheck.sh`’라고 입력한 후 저장하면 된다.

⚠ 참고

‘`crontab -e`’ 명령어를 리눅스 설치 후 최초로 실행하면 텍스트 편집 창으로 전환되기 전 어떤 텍스트 편집기를 사용할 것인지 묻는 문구가 나타난다. 대부분 GNU nano 또는 vi 편집기를 많이 사용한다. 본인이 사용하기 편한 편집기를 선택하면 된다.


```

GNU nano 4.8 /tmp/crontab.7u00qv/crontab
## Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow  command
* * * * * bash ~/runCheck.sh

```

[^]G Get Help [^]O Write Out [^]W Where Is [^]K Cut Text [^]J Justify
[^]X Exit [^]R Read File [^]\ Replace [^]U Paste Text [^]T To Spell

그림 6. crontab 에 영상분석 프로세스 자동 실행 스크립트 등록

E. 프로세스 실행 여부 확인

터미널 창에 ‘`ps -ef | grep run_`’이라고 입력했을 때 [표 16]와 같은 결과가 출력되면 프로세스가 정상적으로 실행 중임을 확인할 수 있다.

```

ves  863620      1 28 06:00 ?          01:35:43 python3 ./run_final.py -
-weights yolov5_topview_v1.pt --conf 0.2 --iou 0.45 --device 0 --nosave
--exist --line 2

```

표 16. 영상분석 프로세스 실행 여부 결과

5. MQTT 를 이용한 영상분석 결과 데이터 수집 (Optional)

안양 2 차 SKV1, 호반파크, 강동역 공영주차장의 영상분석 결과는 MQTT 를 통해 InfluxDB 로 전송된다. 이를 통해 영상분석 정상 작동 여부 또는 영상 데이터 활용 톨을 개발하는데 기반이 될 수 있다. 이 항목에서는 영상분석 결과 데이터를 MQTT 로 전송하는 방법에 대해 설명한다.

A. mqtt 디렉토리

a. MQTTConfigBuilder.py

MQTT 접속 시 사용하는 정보를 호출하는 소스로, MQTTConfigBuilder 클래스로 구성되어 있다. 이 클래스에는 setAndGetUUIDCode, getUUIDCode, getUserConfig 의 세 가지 속성을 포함하고 있다. setAndGetUUIDCode에서는 MQTT 접속 시 사용되는 JSON 파일 (mqttUUID.json)을 작성하여 저장하며, 이 JSON 파일에는 device_id, device_secret, product_key 요소로 구성된다. getUUIDCode 는 MQTT 접속 시 저장한 JSON 파일의 UUID 값을 호출하며, getUserConfig 는 로그인 정보가 포함된 JSON 파일 (watchmileMQTTUser.json)에서 로그인 정보를 호출하여 로그인을 진행한다. MQTTConfigBuilder 클래스 소스코드 전문은 [표 17]에서 확인할 수 있다.

```
class MQTTConfigBuilder:
    def __init__(self):
        mqttPath = MQTTPath()
        self._src = mqttPath.get_file_path("mqttUUID.json")
        self._user_src = mqttPath.get_file_path("watchmileMQTTUser.json")

    def setAndGetUUIDCode(self, uuid):
        new_json = {"device_id": str(uuid), "device_secret": str(uuid),
"product_key": str(uuid)}
        with open(self._src, "w") as json_file:
            json.dump(new_json, json_file)

        return new_json

    def getUUIDCode(self):
        with open(self._src, "r") as json_file:
            auth_json = json.load(json_file)
            json_file.close()

        return auth_json

    def getUserConfig(self, param):
        with open(self._user_src, "r") as json_file:
            user_json = json.load(json_file)
            json_file.close()

        return user_json[param]
```

표 17. MQTTConfigBuilder.py/MQTTConfigBuilder 클래스 소스코드 전문

b. `Registration_MQTT_with_pem_key.py`

장치를 MQTT 서버에 등록하는 소스로, `mqttUUID.json` 및 `watchmileMQTTUser.json` 파일의 정보를 이용해 SSL 을 다운로드하고 장치를 등록한다. 등록할 때는 [표 18]의 소스코드에서 main 함수 내 `builder = MQTTConfigBuilder()`, `builder.setAndGetUUIDCode(uuid.uuid1())` 이 두 줄의 주석을 해제하고 실행한 다음, [그림 7]와 같이 bonfire link 웹페이지 (https://api.wlogs.watchmile.com/wlogs_devices/index)에 접속하여 사용할 장치의 ID 를 'Active' 상태로 전환해야 한다.

My Bonfire

메인 Sign In

all inactive banned deleted organizations ▼						
<input type="checkbox"/>	product_key	device_id	device_platform	device_regions	organizations	connected_on status
<input type="checkbox"/>	asdfsdfasdfsdfasdfsdf	c9b61f22-eba8-11ec-9283-be56f4089b34	embedded		organization_wlogs	--- active
<input type="checkbox"/>	05f9171a-f2d7-11ed-9c88-70cd0ddc3b92	6e4d4bfa-3d9a-4abe-8abc-a2b96d88f00x				--- inactive
<input checked="" type="checkbox"/>	17e30d4a-f2d9-11ec-8253-70cd0ddc3b92	17e30d4a-f2d9-11ec-8253-70cd0ddc3b91				Jul 5, 22 2:22 PM active
<input type="checkbox"/>	32df9320-fc09-11ec-a104-897c036cdebb	32df9320-fc09-11ec-a104-897c036cdebb				Sep 8, 22 10:00 AM active
<input type="checkbox"/>	f56447df-fc0c-11ec-920c-70cd0ddc3b91	f56447df-fc0c-11ec-920c-70cd0ddc3b91				Jul 26, 22 2:48 PM active
<input type="checkbox"/>	cd2f79b0-fc25-11ec-9809-ab2dee6b40dd	cd2f79b0-fc25-11ec-9809-ab2dee6b40dd				Jul 7, 22 2:24 PM active
<input type="checkbox"/>	7676d636-fdbf-11ec-b0bb-e9226336e210	7676d636-fdbf-11ec-b0bb-e9226336e210				Aug 16, 22 11:09 AM active
<input type="checkbox"/>	8879fb4e-0272-11ed-934d-365353a09a82	8879fb4e-0272-11ed-934d-365353a09a82				Sep 8, 22 10:02 AM active
<input type="checkbox"/>	616f0acd-d299-4c5d-bef8-e379f2f83a73	137fcf3c-70b7-4b81-835e-c64518dab3fc				Aug 8, 22 5:04 PM active

With selected activate deactivate ban unBan delete

그림 7. Bonfire link 에서 장치 ID 활성화

⚠ 주의

최초로 장치를 등록하고 영상분석 프로세스를 실행하면 `urlopen error [SSL: CERTIFICATE_VERIFY_FAILED]` 에러가 발생할 수 있다. 그러면 `ssl` 패키지를 import 한 후, `__main__` 함수에 `ssl._create_default_https_context = ssl._create_unverified_context` 구문을 추가하여 `Registration_MQTT_with_pem_key.py` 를 실행한 후 프로세스를 재실행하면 된다.

```

def timeout(i):
    if i <= 1:
        print("Wait for Response : " + str(i) + " second")
    else:
        print("Wait for Response : " + str(i) + " seconds")
    time.sleep(1)

def returnMQTTClient(topic_class, client_id, user_auth, ssl_ft=False):
    mqtt_client = WatchMileMQTT(
        clientID=client_id,
        broker=user_auth['host'],
        port=user_auth['port'],
        user=user_auth['username'],
        pwd=user_auth['password'],
        ssl_check=ssl_ft
    )
    print(ssl_ft)
    return mqtt_client

def registration_mqtt():
    topic_class = "registration"
    config = MQTTConfigBuilder()
    payload = config.getUUIDCode()
    user_auth = config.getUserConfig(topic_class)
    client_id = "wlogs:kor:wlogsORG:embedded-sg20:" + payload['device_id']
    mqtt_client = returnMQTTClient(topic_class, client_id, user_auth, ssl_ft=True)
    mqtt_client.setUser()
    mqtt_client.start()
    reg_topic = "wlogs/device/reg/" + client_id + "/" + topic_class
    mqtt_client.publish(
        topic=reg_topic,
        msg=payload
    )
    mqtt_client.subscribe(topic=reg_topic + "/result")
    print("\nbonfire check 'connected_on'\n")
    ssl_topic = "wlogs/device/reg/" + client_id + "/ssl"
    mqtt_client.publish(
        topic=ssl_topic,
        msg=payload
    )
    mqtt_client.subscribe(ssl_topic + "/result")

    i = 0
    while len(mqtt_client.received_msg.keys()) < 2:
        if i > 10:
            break
        timeout(i)
        i += 1

    mqtt_client.stop()
    return mqtt_client

def registration_pem():
    mqttPath = MQTTPath()
    mqtt_reg = registration_mqtt()
    ssls = json.loads(mqtt_reg.received_msg['ssl'].decode('utf-8'))

    for ssl in ssls.keys():
        if ssl == "ca_cert":
            file_name = "ca_certificate_wlogs.pem"
        elif ssl == "client_cert":
            file_name = "client_mqtt.wlogs.watchmile.com_certificate_wlogs.pem"

```

```

elif ssl == "client_key":
    file_name = "client_mqtt.wlogs.watchmile.com_key_wlogs.pem"
else:
    print("Invalid SSL Keys..")
    break
f = open(mqttPath.get_file_path(file_name), "w") # file_name -> mqttPath.get
f.write(ssls[ssl].replace("\r", ""))
f.close()

if __name__ == "__main__":
    # builder = MQTTConfigBuilder()
    # builder.setAndGetUUIDCode(uuid.uuid1())
    registration_pem()

# bonfire link : https://api.wlogs.watchmile.com/wlogs_devices/index

```

표 18. `Registration_MQTT_with_pem_key.py` 소스코드 일부

c. `Login_MQTT.py`

등록된 MQTT 정보로 로그인하여 토큰을 생성하는 소스코드로, `returnMQTTClient` 함수를 통해 MQTT 클래스를 생성한다. [표 20]에서 소스코드 일부를 확인할 수 있다.

d. `Service_MQTT.py`

`Login_MQTT.py` 에서 출력된 토큰을 비밀번호로 사용하여 재 로그인을 진행하는 소스코드로, `publish_pks_data` 함수를 통해 영상분석 결과 데이터를 MQTT 로 publish 한다. [표 21]에서 `publish_pks_data` 함수의 소스코드 전문을 확인할 수 있다.

B. `run_final.py` 에 적용

`run_final.py` 에 MQTT 전송 기능을 적용하기 위해서는 `run` 함수 내부에 먼저 MQTT 클래스를 정의한 후, 영상분석 완료 후 publish 기능을 구현하면 된다. [표 19]와 같이 기존의 소스코드에 추가하면 MQTT 데이터 전송 기능 구현이 가능하다.

```

.....
detector = Detector(opt)
dataset = LoadData(mode, config_file)
.....
# Image Log Queue Check Service
queueCheckService = QueueCheckService()
# MQTT Client
logins = login_mqtt()
topic_class = 'log'
service_mqtt_client = mqtt_data_client(logins, topic_class)
parkingLot = "skv1"
parkingFloor = "b3"
service_topic = "wlogs/device/service/" + logins.clientID + "/" + topic_class +
"/video/" + parkingLot + "/" + parkingFloor
start = time.time()

# Run inference
detector.model_warmup(bs) # warmup
.....
result = insert_data(algorithm, cam_seq, cam, boxes, imgpath)
.....
# MQTT publish

```

```

        if (time.time() - start) / 60 > 30:
            cp = check_token_is_expired(service_mqtt_client)
            service_mqtt_client.password = cp
            service_mqtt_client.stop()
            service_mqtt_client.setUser(pwd=cp)
            service_mqtt_client.start()
            start = time.time()
        publish_pks_data(service_mqtt_client, service_topic,
            queueCheckService.getterResult(), parkingFloor)

```

표 19. run_final.py 에 MQTT 전송 기능 구현

```

def returnMQTTClient(topic_class, client_id, user_auth, ssl_ft=False):
    mqtt_client = WatchMileMQTT(
        clientID=client_id,
        broker=user_auth['host'],
        port=user_auth['port'],
        user=user_auth['username'],
        pwd=user_auth['password'],
        ssl_check=ssl_ft
    )
    return mqtt_client

def login_mqtt():
    mqttPath = MQTTPath()
    topic_class = "login"
    config = MQTTConfigBuilder()
    payload = config.getUUIDCode()
    user_auth = config.getUserConfig(topic_class)
    client_id = "wlogs:kor:wlogsORG:embedded-sg20:" + payload['device_id']
    mqtt_client = returnMQTTClient(topic_class, client_id, user_auth, ssl_ft=True)
    mqtt_client.setUser()
    mqtt_client.setssl()
    mqtt_client.start()
    login_topic = "wlogs/device/auth/" + client_id + "/" + topic_class
    mqtt_client.publish(
        topic=login_topic,
        msg=payload
    )
    mqtt_client.subscribe(topic=login_topic + "/result")

    i = 0
    while len(mqtt_client.received_msg.keys()) < 1:
        if i > 20:
            break
        timeout(i)
        i += 1

    mqtt_client.stop()
    return mqtt_client

```

표 20. Login_MQTT.py 소스코드 일부

```

def publish_pks_data(client, topic, result, parkingFloor):
    count = len(result['pks-id-list'])
    result_list = []
    for i in range(count):
        created = datetime.datetime.now().timestamp()
        updated = datetime.datetime.now().timestamp()
        return_params = {
            "region": result['region'],
            "pkName": result['pk-name'],
            "pkFloor": parkingFloor,
            "cctvId": result['cctv-id'],
            "pksId": result['pks-id-list'][i],
            "ROIBoxCoord": ",".join(map(str, result['roi-box-coord-list'][i])),
            "predictedByPerson": "",
            "modelName": result['model-name'],
            "modelVersion": result['model-version'],
            "created": created,
            "updated": updated,
            "indoorDivision": "",
            "weatherForecast": ""
        }
        if result['img-path']:
            return_params['imgPath'] = result['img-path']
        if result['img-base64']:
            return_params['imgBase64'] = str(result['img-base64'])
        if result['predicted-by-model-list']:
            if result['predicted-by-model-list'][i] == 'free':
                return_params['predictedByModel'] = 0
            else:
                return_params['predictedByModel'] = 1
        if result['predicted-by-model-list-v1']:
            if result['predicted-by-model-list-v1'][i] == 'free':
                return_params['predictedByModelV1'] = 0
            else:
                return_params['predictedByModelV1'] = 1
        if result['predicted-by-model-list-v2']:
            if result['predicted-by-model-list-v2'][i] == 'free':
                return_params['predictedByModelV2'] = 0
            else:
                return_params['predictedByModelV2'] = 1
        if result['bounding-box-coord-list']:
            return_params['boundingBoxCoordList'] = ",".join(
                ["",".join(map(str, s)) for s in result['bounding-box-coord-list']]
            )
        result_list.append(return_params)
    client.publish(topic, {"pkslog": result_list})

```

표 21. `Service_MQTT.py/publish_pks_data` 함수 소스코드 전문

실내 주차장 내 보행자 인식 (Pedestrian2Map)

1. 개요

주차장 내의 여러 CCTV를 이용하여 실시간으로 보행자를 검출하고 검출한 보행자 위치를 지도 위에 표시하여 자율주행 자동차의 운행 시스템에 전달한다. 이를 통해 자율주행 자동차는 주행 전에 주차장 내에 보행자 위치를 인식하고 사전에 보행자에게 가해질 위험을 최소화할 수 있게 있도록 하는 개발 프로젝트이다.

A. 개발 환경

보행자 인식 알고리즘을 구현하기 위한 서버의 하드웨어 스펙 및 필수 패키지 목록을 [표 22]와 [표 23]로 정리했다.

OS	CPU	RAM	GPU
Linux Mint 20.2	AMD Ryzen 5 5600X	32GB	RTX 3080Ti 12GB

표 22. 보행자 인식용 영상분석 서버 하드웨어 사양

```
# pip install -r requirements.txt

# Base -----
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# Logging -----
tensorboard>=2.4.1

# wandb

# Plotting -----
pandas>=1.1.4
```



```
seaborn>=0.11.0
```

```
thop # FLOPs computation
```

표 23. 필수 설치 라이브러리 목록 (`requirements.txt`)

B. Homography

Yolov5 모델은 실시간 영상을 통해서 보행자를 인식한다. 이 때 사용되는 학습 가중치는 모델이 기본적으로 제공하는 가중치 (Yolov5s, Yolov5m, Yolov5l, Yolov5x)를 사용한다. 그 후 보행자를 인식한 결과를 이용하여 지도 위에 표시하기 위해서 Homography 를 이용한다. Homography 란 [그림 8]와 같이 한 평면을 다른 평면에 투영시켰을 때 대응되는 점들 사이에 대응되는 일정한 변환 관계를 의미한다. 이를 이용하여 각 영상에서 표현되는 좌표와 지도에서의 좌표 값을 대응하여 변환 관계를 찾아 보행자 위치를 지도 위에 표시했다. 23 개의 CCTV 와 하나의 지도에 대해 좌표 값을 매칭하여 각 영상 별로 변환 관계를 계산하였으며, 영상 별 변환 행렬은 `config_hd_2.py` 에 정리되어 있다.

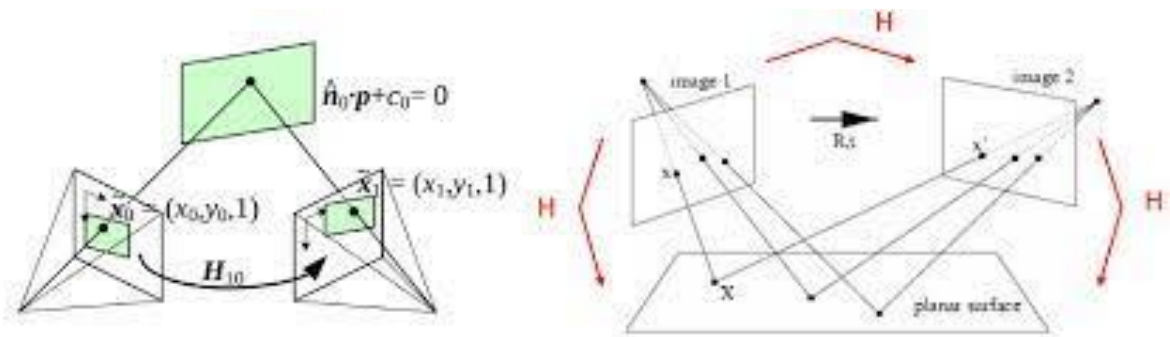


그림 8. Homography 의 개념

2. 보행자 인식 소스코드 설명

A. util/config_hd_2.py

`config_hd_2.py` 안에는 각 CCTV 별로 필요한 정보들을 저장하고 있다. `cams` 내에 각각 CCTV 이름으로 key 값이 존재하고 key 값에는 CCTV 주소를 포함하는 'src' 및 CCTV 별 Homography 변환 행렬을 포함하는 'homoMat'이 존재한다. 여기서는 두 값을 이용하여 CCTV에 접속하고 CCTV에서 검출한 보행자 위치를 변환 행렬을 통해서 변환한다. [표 24]와 같이 'homoMat' 값을 설정할 수 있다.

```
cams = {
    'CCTV02': {
        'road': [[62, 688], [7, 684], [4, 366], [393, 310], [591, 105], [966, 97],
        [1033, 266], [1295, 296],
        [1914, 354], [1915, 620], [1470, 668], [1734, 1073], [20, 1062]],
        'src': "rtsp://admin:123456@172.10.14.12/media/video1",
        'uuid': [
            "a68712cb-0a2b-11ec-9c5c-4616e5ba6420",
            "a6b65ad6-0a2b-11ec-9c5c-4616e5ba6420",
            "a6b3847b-0a2b-11ec-9c5c-4616e5ba6420",
        ],
        'pos': [
            "1-y2-x6-1",
            "1-y3-x5-2",
            "1-y3-x5-1"
        ],
        'roi': [[648, 116, 775, 180], [850, 165, 950, 232], [1035, 174, 1167, 246]],
        'homoMat': [[-0.3073809669587126, 7.99707279067324, 2702.7232410911593],
        [0.11904321736998807, 4.8732445695710584, 2445.109893443374],
        [8.205149899555664e-05, 0.003500582969557371, 1.0]]
    },
}
```

표 24. `config_hd_2.py` 일부

B. main_seq.py

a. getFrame

소스코드 내에서 `main.py`와 `main_seq.py`가 존재하는데 여기서는 `main_seq.py`를 다룬다. 그 이유는 최초 개발 시 실시간성을 위해서 CCTV에서 영상을 얻어 Yolo를 통해 보행자를 인식하는 것까지 하나의 프로세스로 생성해서 실행했기 때문이다. 그러나 여기서는 사용할 수 있는 GPU가 하나이기에 프로세스가 증가함에 따라서 GPU 메모리 부족을 초래한다. 그래서 영상에서 프레임만 병렬적으로 가져오고 추론은 순차적으로 실행함으로써 GPU 메모리 부족 문제를 해결하였다. 따라서, 소스코드 이름에 `seq`가 붙어 있다.

그래서 `getFrame`에 의해 CCTV 영상 수만큼 프로세스를 생성해서 병렬적으로 계속 프레임을 가져온다. 그 후 `return_dict['img']`['저장하고자 하는 CCTV 이름']안에 프레임을 저장한다. `return_dict`는 공유 변수로서 다른 프로세스들도 공유하는 변수이다. 이를 통해서 계속해서 새로운 프레임을 저장하여 실시간성 있는 추론을 가능하도록 한다. 또한, `getFrame`에서는 자신에게 주어진 변수에만 접근을 하기

때문에 비동기적으로 실행하더라도 문제가 발생하지 않는다. [표 25]에서 이 함수의 소스코드 전문을 확인할 수 있다.

```
def getFrame(cctv_addr, cctv_name, return_dict):
    font = cv2.FONT_HERSHEY_SIMPLEX # 글씨 폰트
    cap = cv2.VideoCapture(cctv_addr)
    while True:
        # start_time = time.time()
        ret, frame = cap.read()
        # end_time = time.time()
        # return_dict['FRAME_TIME'] = end_time-start_time
        # print(f'{cctv_name} 프레임 가져오는 시간 - {round(end_time - start_time, 3)} s')
        if ret:
            return_dict['img'][cctv_name] = frame
        else:
            Error_image = np.zeros((720, 1920, 3), np.uint8)
            cv2.putText(Error_image, "Video Not Found!", (20, 70), font, 1, (0, 0, 255),
3) # 비디오 접속 끊어짐 표시
            return_dict['img'][cctv_name] = Error_image

        # reconnect
        cap.release()
        cap = cv2.VideoCapture(cctv_addr)
```

표 25. `getFrame` 함수 소스코드 전문

b. `detect`

본 개발 코드에서 제일 핵심이 되는 함수이다. 앞서 말한 것과 같이 CCTV 영상은 병렬적으로 가져오지만 추론은 순차적으로 한다고 언급하였다. 위에 코드에서 보다시피 PyTorch를 통해 로드된 Yolo 모델 안에 `img`가 순차적으로 입력됨을 확인할 수 있다. 그러나 입력되는 `img`는 추론되기 전까지는 계속해서 최신화 되어 입력된다. 그 이유는 병렬적으로 이미지를 가져오는 프로세스들이 `return_dict` 내에 계속해서 저장하고 있고 `detect`는 카메라별로 순차적으로 추론하지만 그 때 입력되는 이미지들은 `return_dict` 안에 저장된 이미지를 사용하기 때문에 과거의 이미지를 사용하지 않는다. 그래서 순차적으로 실행되더라도 최대한 실시간성을 보장할 수 있도록 했다. 그 후 Yolo에 의해 보행자를 인식하면 보행자를 인식한 bounding box 하단 중심점을 homography를 사용하여 지도에 표시할 좌표 값으로 변환한다. 그래서 이 값을 `return_dict`에 저장하여 구해진 모든 좌표 정보를 지도 위에 표시하고 서버로 전송한다. [표 26]에서 이 함수의 소스코드 전문을 확인할 수 있다.

```

def detect(return_dict):
    font = cv2.FONT_HERSHEY_SIMPLEX # 글씨 폰트
    # yolov5
    # 로컬 레포에서 모델 로드(yolov5s.pt 가중치 사용, 추후 학습후 path에 변경할 가중치 경로 입력)
    # 깃허브에서 yolov5 레포에서 모델 로드
    # model = torch.hub.load('ultralytics/yolov5', 'custom',
    path='yolov5s.pt', device=num%3)
    model = torch.hub.load('yolov5', 'custom', path='yolov5s.pt', source='local',
    device=0)
    # 검출하고자 하는 객체는 사람이기 때문에 coco data에서 검출할 객체를 사람으로만
특정(yolov5s.pt 사용시)
    model.classes = [0]
    model.conf = 0.5
    window_width = 320
    window_height = 270
    # CCTV 화면 정렬
    for num, cctv_name in enumerate(cams.keys()):
        cv2.namedWindow(cctv_name)
        cv2.moveWindow(cctv_name, window_width * (num % 6), window_height * (num // 6))

    # CCTV 화면 추론
    while True:
        for cam_index, cctv_name in enumerate(cams.keys()):
            # 추론
            img = return_dict['img'][cctv_name]
            # yolo_start_time=time.time()
            bodys = model(img, size=640)
            # yolo_end_time=time.time()
            # return_dict['YOLO_TIME'] = yolo_end_time-yolo_start_time
            # print(f'yolov5 {cctv_name} img 추론 시간 - {round(end_time - start_time, 3)}
s')

            flag = False
            points = []

            # yolo5
            # homo_start_time = time.time()
            for i in bodys.pandas().xyxy[0].values.tolist():
                # 결과
                x1, y1, x2, y2, conf, cls, name = int(i[0]), int(i[1]), int(i[2]), int(i[3]),
i[4], i[5], i[6]

                cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2) # bounding box
                cv2.putText(img, name, (x1 - 5, y1 - 5), font, 0.5, (255, 0, 0), 1) #
class 이름
                cv2.putText(img, "{:.2f}".format(conf), (x1 + 5, y1 - 5), font, 0.5, (255, 0,
0), 1) # 정확도

            # 보행자 좌표 표시
            target_x = int((x1 + x2) / 2) # 보행자 중심 x 좌표
            target_y = int(y2) # 보행자 하단 좌표

            # 보행자 픽셀 위치 표시
            img = cv2.circle(img, (target_x, target_y), 10, (255, 0, 0), -1)
            cv2.putText(img, "X:{} y:{}".format(target_x + 5, target_y + 5),
(target_x + 10, target_y + 10), font,
0.5, (255, 0, 255), 1)

```

```

        # homography 변환
        target_point = np.array([target_x, target_y, 1], dtype=int)
        target_point = target_point.T
        H = np.array(cams[cctv_name]['homoMat'])
        target_point = H @ target_point
        target_point = target_point / target_point[2]
        target_point = list(target_point)
        target_point[0] = round(int(target_point[0]), 0) # x -> left
        target_point[1] = round(int(target_point[1]), 0) # y -> top
        points.append((target_point[0], target_point[1]))

        flag = True # 변환된 정보 저장

# homo_end_time = time.time()
# return_dict['HOMOGRAPHY_TIME'] = homo_end_time-homo_start_time
# 변환된 보행자 픽셀 위치 저장
if flag:
    return_dict[cctv_name] = (flag, points)
else:
    return_dict[cctv_name] = (False, [])
    temp_img = cv2.resize(img, dsize=(window_width, window_height))
    cv2.imshow(cctv_name, temp_img)
    send2server(return_dict)
    k = cv2.waitKey(1) & 0xff
    if k == 27:
        break

```

표 26. **detect** 함수 소스코드 전문

c. 조건별 성능 분석

현재 개발된 알고리즘을 통해서 한 영상을 인식하는데 전체적으로 걸리는 시간을 계산하면 아래의 수식과 같다.

$$T_{total} = T_{cctv} + T_{server} + \sum_{k=1}^n T_{yolo_k}$$

하나의 CCTV 정보를 전달한다고 가정하였을 때 T_{cctv} 에 대한 시간만 필요로 하다. T_{server} 는 종합적으로 모든 데이터는 MQTT 방식으로 서버로 전송할 때 소요되는 시간이다. T_{yolo} 는 Yolo에 의해 소요되는 시간이고 모든 CCTV에 대해서 추론을 해야 하므로 각각 CCTV 수에 따라 시간이 커지게 된다. 그래서 전체 시간은 CCTV에 의해 크게 의존적이라고 판단할 수 있다. 그래서 CCTV 변화에 따라 성능을 평가하였다. CCTV 영상은 RTSP 방식을 통해서 가져오게 된다. RTSP는 TCP 방식을 사용하기 때문에 연결 상태를 계속해서 확인해야 하므로 여러 영상에 동시에 접속할 때 약간의 지연이 발생할 수 있다. 그래서 이 문제를 해결하기 위해 컴퓨터를 CCTV와 같은 네트워크 상에 설치한 후 물리적인 지연을 최소화하였으며, 기존에 다른 서버를 거쳐 영상을 가져올 때에 비해서 빠른 영상 읽기가 가능하였다. 그럼에도 불구하고 CCTV 지연이 가장 큰 문제라고 판단되어 CCTV 영상 수에 따른 성능 평가를 실시했으며 그 결과는 [표 27]에 기록하였다.

CCTV 대수	영상 가져오기 (중앙값) [s]	영상 가져오기 (최대값) [s]	YOLO 추론 시간 (중앙값) [s]	YOLO 추론 시간 (최대값) [s]
1	0.025	0.102	0.009	0.013
5	0.039	0.286	0.010	0.017
10	0.028	30.039	0.009	0.030
15	0.004	0.478	0.011	0.024
23	0.004	0.792	0.014	0.029

표 27. CCTV 대수에 따른 처리 시간 변화

성능을 평가하기 위한 시간 측정은 평균 값이 아닌 중앙 값을 사용하였다. 평균 값을 사용하면 비정상적으로 너무 긴 시간에 의해서 평균 시간이 너무 길어질 수 있기 때문이다. 따라서, 중앙 값으로 CCTV 개수에 따른 시간 측정을 비교하였다. 비교 결과 CCTV 영상 가져오는데 연결된 CCTV 대수가 증가함에 따라서 중앙 값으로는 더욱 짧은 처리 시간을 보였다. 또한, YOLO 같은 경우 중앙 값과 최대 값을 비교하였을 때 최대 0.02 초 정도의 차이만 보였다. 그러나 영상을 가져오는 부분에서 소요되는 시간이 최대 30 초까지 걸리는 문제를 발견하였다. 모델이 보행자를 인식하는데 최대 0.03 초가 소요되는 반면 영상 자체를 가지고 오는 것에 지연이 되는 시간이 너무 크다는 것을 알 수 있었다. 즉, 실시간으로 영상을 처리하기 위해서는 안정적인 영상 스트리밍이 관건이라고 확인했다.

C. test_save_photo.py - getframe

거듭 실행해본 결과 또다른 문제점을 발견할 수 있었다. 멀티프로세싱을 이용해 병렬로 실행할 때 각 프로세스들은 원래 공유하는 변수가 존재할 수 없다. 그러나, Manager 안에 있는 return_dict 를 사용하여 각 프로세스들이 이미지 파일을 공유할 수 있도록 하였는데 실제로 설정된 변수에 Global Interpretation Lock (GIL) 기능이 포함되어 있어 읽는 것이 더욱 느려지는 문제점을 발견할 수 있었다. 그래서, 코드 내에서 공유 변수를 사용하지 않고 CCTV 들이 얻은 이미지를 파일로 저장하여 YOLO 에서 꺼내서 쓸 수 있도록 변경하였다. 그 결과 확연히 기존보다 실시간성을 얻었지만, [그림 9]과 같이 간혹 이미지를 정확히 가져오지 못하는 문제점이 발생하였다.

[표 28]에서와 같이 기존에는 return_dict 를 통해서 이미지 파일을 저장하였지만, 여기서는 OpenCV 의 imwrite 함수를 사용하여 이미지 파일로 저장을 한다. 그리고 계속해서 파일을 덮어쓰므로써 공유변수 역할을 대신 수행하게 하였다.

마찬가지로 detect 함수 내부에서도 파일을 return_dict 가 아니라 OpenCV 의 imread 함수를 이용해 storage 에서 파일을 가져와 이미지를 처리하도록 하였다.

일반적으로 RAM 보다 Storage 에서 데이터를 저장하고 쓰는 것이 느린 것이 사실이다. 그러나 본 개발에서 서버와 CCTV 간의 물리적 거리를 고려하여 소프트웨어적으로 공유변수를 사용하였지만, 프로세스간 lock 권한을 acquire 하는 과정에서 지연이

더욱 크게 발생하였다. 특히, 본 개발에서 데이터 간에 critical zone 은 존재하지만 서로 영향을 주지 않기 때문에 불필요한 lock 이 오히려 성능을 저하시키는 것을 확인했다. 그래서 물리적 거리가 멀어지더라도 불필요한 lock 을 줄임으로서 프로세스 수행 시간을 단축시킬 수 있었다.

```
def getFrame(cctv_addr, cctv_name):
    font = cv2.FONT_HERSHEY_SIMPLEX # 글씨 폰트
    cap = cv2.VideoCapture(cctv_addr)
    cap.set(cv2.CAP_PROP_XI_RECENT_FRAME, 1)
    cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
    while True:
        # start_time = time.time()
        ret, frame = cap.read()
        # end_time = time.time()
        # return_dict['FRAME_TIME'] = end_time - start_time
        # print(f'{cctv_name} 프레임 가져오는 시간 - {round(end_time - start_time, 3)} s')
        if ret:
            cv2.imwrite("./img/" + cctv_name + ".jpg", frame)
        else:
            Error_image = np.zeros((720, 1920, 3), np.uint8)
            cv2.putText(Error_image, "Video Not Found!", (20, 70), font, 1, (0, 0, 255),
3) # 비디오 접속 끊어짐 표시
            cv2.imwrite("./img/" + cctv_name + ".jpg", Error_image)

            # reconnect
            cap.release()
            cap = cv2.VideoCapture(cctv_addr)
```

표 28. test_save_photo.py - getframe 함수 소스코드 전문

```
for cam_index, cctv_name in enumerate(cams.keys()):
    # 추론
    try:
        img = cv2.imread("./img/" + cctv_name + ".jpg")
    except:
        continue
```

표 29. test_save_photo.py 내 이미지 저장 부분 소스코드

3. 고찰

보행자를 인식하고 이를 지도상에 표시하기 위해서 YoloV5 와 homography 를 사용하였다. 그리고 실시간성을 위해서 영상을 읽고 이를 추론하는 과정을 병렬적으로 실행하여 최대한 실시간성을 보장하였다. 또한 이 결과를 서버로 전송하여 앱에 표시하는 것까지 모두 완료했으며 [그림 10]과 같이 비교적 원활히 동작함을 확인할 수 있었다. 그러나 인식의 안정성 측면에 대해서는 조금 더 관찰할 필요가 있다. 게다가 기존에 실시간성 향상을 위해 Yolo 의 추론 시간이 느리다고 판단하여 TensorRT 프레임워크를 사용하는 등 영상분석 속도를 가속화하여 최대한 시간을 단축시키고자 했다. 그런데 성능을 평가하면서 Yolo 의 영상분석 시간보다는 CCTV 를 스트리밍하는 시간이 가장 큰 bottleneck 임을 수치로 확인할 수 있었다. 따라서 추후

원활한 실시간성을 보장하기 위해서는 영상 스트리밍 지연 시간을 최소화하는 것이 필요하다.

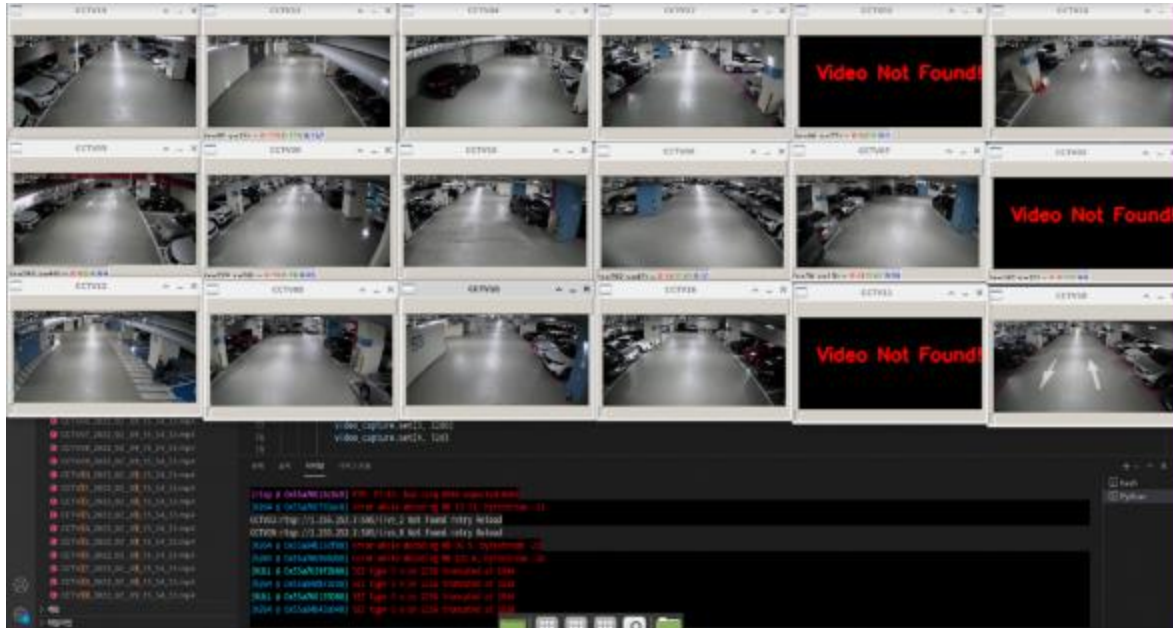


그림 9. 다중 CCTV 스트리밍 후 보행자 인식 실행 결과

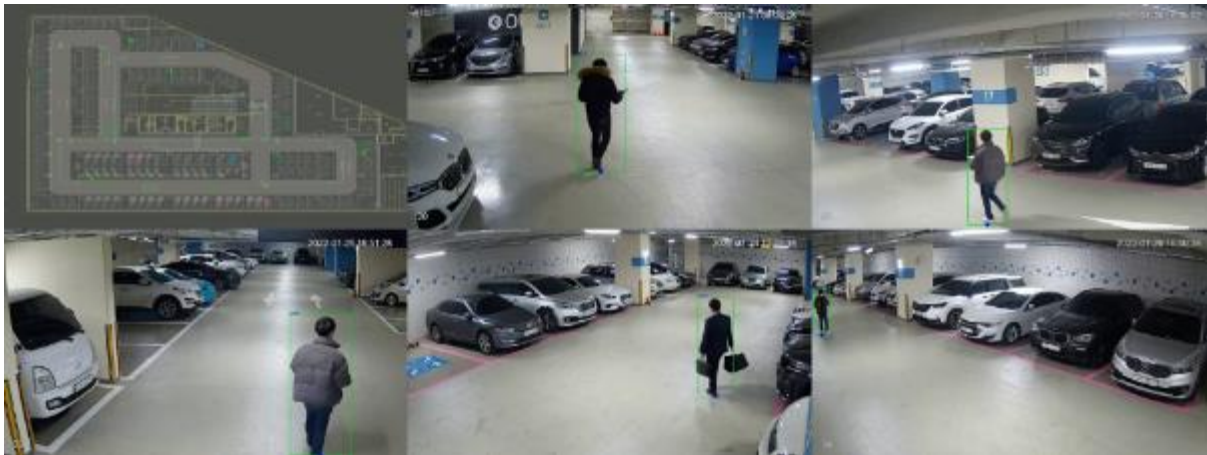


그림 10. 실제 보행자 인식 결과

4. 실행 방법

모든 파일들은 `util` 안에 있는 `config_hd_2.py`의 '`homoMat`'값을 잘 정의한 후 `python *.py`로 실행하면 된다. 또한, `util` 폴더 안에는 homography 행렬을 계산해주는 `computeMat.py`와 이미지 픽셀 정보를 추출해주는 `pixelpicker.py`가 있으니 이를 사용해도 되고 스스로 찾아서 사용해도 좋다. 이 코드들은 실제로 실행하는데 필수적인 코드들이 아니라 homography 행렬 값을 구하기 위해서 사용하는 코드이기 때문에 실제 실행에는 의존적이지 않다.

실시간 장애물 위치 정보 제공 툴 (ClickPedestrian)

1. 개요

본 툴은 보행자 인식 정보를 MQTT 방식으로 정보를 전송하여 가상으로 보행자 위치를 표시할 수 있게 해주는 툴이다. 이를 통해서 YOLO 를 통한 보행자 인식이 원활하지 않을 경우 본 툴을 통해서 임의로 정보를 전송할 수 있다.

2. 필수 라이브러리

`opencv-python` / `paho-mqtt`

위 두 라이브러리만 설치되더라도 모든 곳에서 사용할 수 있는 툴이다. 위 라이브러리를 일일이 설치하면 된다. 한 번에 설치를 원한다면 `requirements.txt` 에 정의되어 있으니 `pip3 install -r requirements.txt` 을 터미널에 입력하면 된다.

3. 실행 방법

\$ `python main.py`

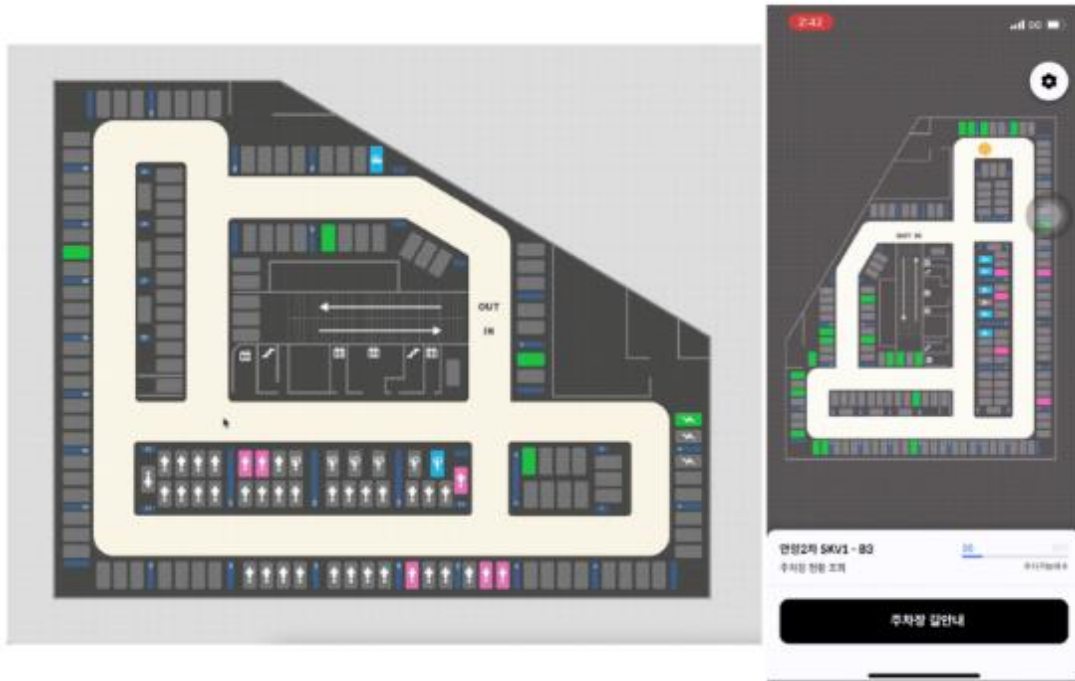


그림 11. 실행 결과

본 개발의 경우 안양 2 차 SKV1 지하 3 층에 맞추어져 있으며, 추후 MQTT 서버 IP 주소가 변경된다면, `util` 디렉토리 내에 있는 `transmit_server.py` 에서 `topic`, `host`, `port` 의 값을 변경하면 된다.

실내 주차장 내 차량 Tracking (Car_Tracking)

1. 개요

본 개발은 안양 2 차 SKV1 주차장을 기준으로 차량을 tracking 하는 개발이다. 본 개발의 시초는 21 년 여름 강승훈 인턴이 천호역 주차장에서 차량 tracking 을 테스트한 결과에서 유래했으며 본 개발에서는 이를 바탕으로 조금 더 개선했다. 따라서 강승훈 인턴이 차량 tracking 을 위해 사용했던 stitching 알고리즘을 재사용하여 개발하였다.

2. stitching 알고리즘

stitching 알고리즘이란 다른 CCTV 에서 얻어진 좌표를 통해 각 좌표들의 거리를 측정하여 두 점의 좌표가 특정 거리 이하이면 같은 객체라고 판단하고 tracking 을 하는 것이다. [표 31]에서 stitching 알고리즘의 소스코드 전문을 확인할 수 있다.

A. 추가 사항

차량 tracking 을 위해서는 도로에만 있는 차량을 검출할 수 있어야 한다. 그래서 config_hd_2.py 의 homography 값과 더불어 road 라는 도로 영역을 표시하는 다각형 좌표 값이 있다. 그래서 이 값을 이용해 OpenCV 의 pointPolygonTest 함수로 특정 좌표가 다각형 내부에 위치한지 외부에 위치한지를 판단하여 도로에 있는 차량에 대해서만 tracking 을 한다. 이 과정은 [표 30]의 소스코드로 구현하고 있다.

```
# 특정 구역에서만 Object 표시
black_img = np.zeros((h, w, c), dtype=np.uint8)
road_poly = np.array(cams[cctv_name]['road'])
black_img = cv2.fillPoly(black_img, [road_poly], (255, 255, 255))
black_img = cv2.cvtColor(black_img, cv2.COLOR_BGR2GRAY)
res, thr = cv2.threshold(black_img, 127, 255, cv2.THRESH_BINARY)
contours, his = cv2.findContours(thr, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
cnt = contours[0]

# 차량이 도로안에 있는 것만 검출
if cv2.pointPolygonTest(cnt, (target_x, center_y),
                        False) > -1: # 도로로 표시한 polygon 안에 있는 경우에만 검출 차량이
    도로에 있으면 1 없으면 -1 정확히 겹치면 0
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2) # bounding box
    cv2.putText(img, name, (x1 - 5, y1 - 5), font, 0.5, (255, 0, 0), 1) # class 이름
    cv2.putText(img, "{:.2f}".format(conf), (x1 + 5, y1 - 5), font, 0.5, (255, 0, 0), 1)
# 정확도

# 보행자 픽셀 위치 표시
img = cv2.circle(img, (target_x, target_y), 10, (255, 0, 0), -1)
cv2.putText(img, "X:{ } y:{ }".format(target_x + 5, target_y + 5), (target_x + 10,
target_y + 10), font, 0.5, (255, 0, 255), 1)
```

표 30. 도로 내부에 위치한 차량 확인용 소스코드 일부

```

# 동일한 프레임 시간대에 지도에 표시되는 모든 좌표 저장
for cctv_name in cams.keys(): # 모든 CCTV 에서 좌표 가져오기
    flag, points = data[cctv_name] # 현재 CCTV 에서 좌표 정보가 있는지
    # 초기 모든 지도 좌표 저장
    for (pX, pY) in points:
        all_temp_points.append((pX, pY))

# 추론된 좌표들 중에서 이미 같은 것이라고 판단된 좌표들은 삭제
for (aX, aY) in all_temp_points:
    for (tX, tY) in temp_points:
        if finddistance(aX, aY, tX, tY) < threshold_dist: # 설정한 거리보다 가까우면 이미
            # 포함된 좌표라고 판단
            break
    else:
        temp_points.append((aX, aY)) # 기존에 없던 새로운 좌표

# 이전 좌표들과 최대한 가까운 좌표 검출
for (tX, tY) in temp_points:
    Min = 1e9 # 최소값을 찾기 위한 초기화
    Min_car_index = 0 # 유사성이 높은 차량 번호
    similar_flag = 0 # 유사성이 높은 차량이 존재여부 FLAG

    # 이전 10 프레임 좌표들과 비교하였을 때 가장 비슷한 좌표 찾기 -> 이를 통해 같은 차량이라고 판단
    track_points = list()
    for num in range(9):
        track_points.extend(track_point[num])
    for (car_index, prevX, prevY) in track_points:
        dist = finddistance(tX, tY, prevX, prevY) # 이전 좌표와 현재 좌표거리들을 비교
        if dist < threshold_dist: # 좌표거리가 특정 거리 이하면은 판단
            if Min > dist: # 특정 거리 이하 중에 제일 가까운 좌표
                similar_flag = True # 유사성이 있는 좌표가 있는 것으로 판단
                Min = dist # 가장 유사성이 높은 좌표 거리 저장
                Min_car_index = car_index # 유사성이 높은 차량 index 번호 저장

    if similar_flag:
        temp_trackpoints.append((Min_car_index, tX, tY)) # 유사성이 있었으니 기존 index 에
        # 추가
    else:
        temp_trackpoints.append((new_car_index + 1, tX, tY)) # 유사성 있는 것이 없었으니
        # 새로운 값으로 할당
    new_car_index += 1 # 새로운 차량 추가

```

표 31. stitching 알고리즘 코드

3. 실행 방법

필요한 모듈 설치

```
$ pip3 install -r requirements.txt
```

tracking 시작

```
$ python3 main.py
```

4. main_seq.py

수많은 CCTV 를 연결해야 하므로 프로세스를 병렬적으로 실행을 하되 메모리 사용은 최소한으로 해야 한다. 그래서 데이터를 계속 한 곳에 뒹어쓰므로써 사용되는 메모리를 줄였으며 현재 24 개의 영상에 대해서는 문제없이 동작을 확인하였다. (이는 기존 보행자 인식 소스코드에서 참고하여 개발했다.) 실행 방식은 보행자 인식 소스코드와 동일하여 설명은 생략한다.

5. 실행 결과

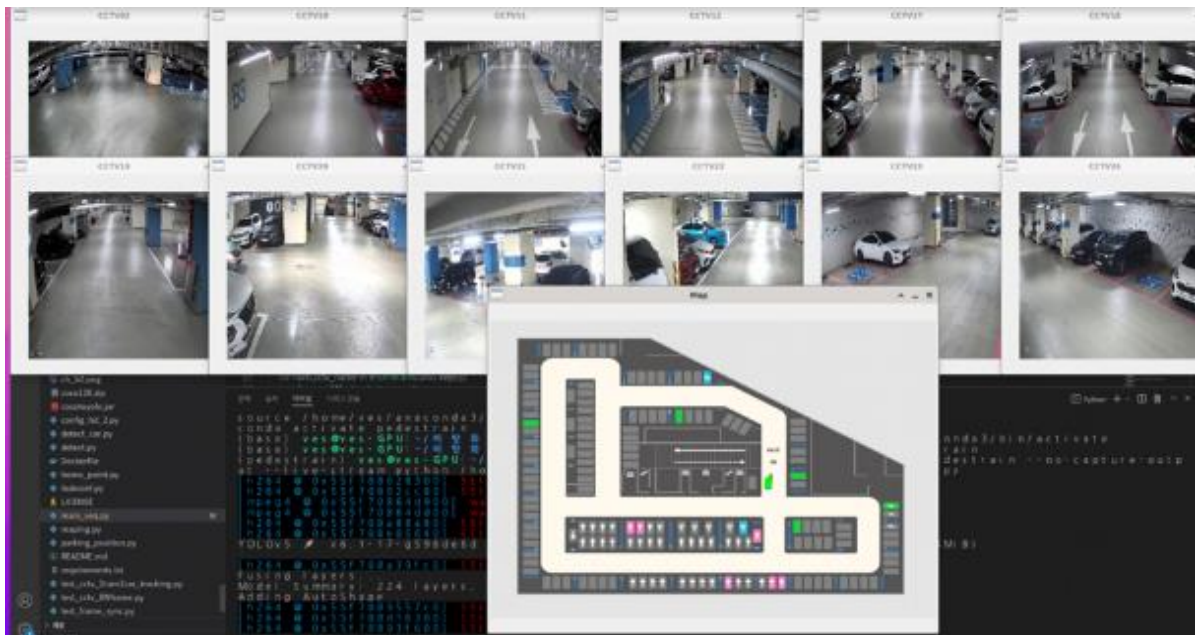


그림 12. 차량 tracking 실행 모습



그림 13. 실행한 후 tracking 결과

6. 문제점

본 개발을 통해 겪었던 문제점은 모든 카메라의 싱크가 맞지 않는다는 것이다. 그렇기에 동일한 장소를 비추는 장소임에도 불구하고 카메라마다 보이는 위치가 달라지는 것이다. 또한 주차 중인 차량이 도중에 출차하는 경우도 고려하였을 때 일정 이상 거리가 가까운 차량을 하나의 차량으로 간주하는 것은 tracking 시 문제가 발생한다. 그렇지만 싱크가 맞으면 stitching 알고리즘으로 단독 객체에 대한 tracking은 가능함을 확인하였다. 그러나 다중 객체의 tracking을 위해서는 stitching 알고리즘이 아니라 다른 CCTV 화면에서도 정확히 객체를 구별할 수 있는 방법을 고안할 필요가 있다.

친환경 차량 검출 및 번호판 OCR (ALPR)

1. 개요

ALPR은 Automatic Number-Plate Recognition의 약자로서 광학으로 자동차 번호판을 읽고 데이터를 생성하는 기술이다. 본 개발에서는 단순히 번호판을 읽고 데이터화 하는 것뿐만 아니라 차량 번호판의 색상을 통해서 해당 차량이 친환경 차량인지 아닌지 구분하는 역할까지 수행한다. 이를 위해서 Yolo를 통해서 차량에서 번호판을 검출한 후 검출한 번호판의 색상을 비교하고 OCR을 통해 번호판을 읽어 데이터를 얻도록 하였다.

2. 개발 환경

본 개발의 핵심 라이브러리는 Yolo 필수 라이브러리 및 easyocr 모듈이다.

```
$ pip install -r requirements.txt
```

위 명령어를 통해서 [표 32]에 나와있는 필요한 라이브러리 모두를 설치할 수 있다.

```
# pip install -r requirements.txt

# Easy OCR -----

easy-ocr

# Base -----

matplotlib>=3.2.2

numpy>=1.18.5

opencv-python>=4.1.2

Pillow>=7.1.2

PyYAML>=5.3.1

requests>=2.23.0

scipy>=1.4.1

torch>=1.7.0

torchvision>=0.8.1

tqdm>=4.41.0

# Logging -----
```

```

tensorboard>=2.4.1

# wandb

# Plotting -----

pandas>=1.1.4

seaborn>=0.11.0

# Export -----

# coremltools>=4.1 # CoreML export

# onnx>=1.9.0 # ONNX export

# onnx-simplifier>=0.3.6 # ONNX simplifier

# scikit-learn==0.19.2 # CoreML quantization

# tensorflow>=2.4.1 # TFLite export

# tensorflowjs>=3.9.0 # TF.js export

# opencv-dev # OpenVINO export

# Extras -----

# albumentations>=1.0.3

# Cython # for pycocotools https://github.com/cocodataset/cocoapi/issues/172

# pycocotools>=2.0 # COCO mAP

# roboflow

thop # FLOPs computation

```

표 32. ALPR 용 설치 패키지 리스트 파일 ([requirements.txt](#))

이 때, easyocr 패키지의 경우 OpenCV와 충돌이 발생할 수 있고, 또한 GPU를 통한 OCR을 사용하기 위해서는 CUDA와 CUDA Toolkit 버전 일치가 매우 중요하다. CPU를 통해서도 가능한 하지만, 빠른 OCR이 불가능하다.

3. util 디렉토리

A. KakaoOCR.py

카카오 Vision API 를 사용해서 OCR 을 하는 모듈이다. 한글 인식률이 뛰어나며 정확도면에서도 다른 tesseract 보다는 우수하다. 그러나 네트워크를 통해서 OCR 을 하기 때문에 느리다는 단점이 있다. 또한, 번호판보다는 전체적인 이미지를 보냈을 때 OCR 이 원활하게 된다. 그 이유는 카카오 OCR 은 이미지에서 글자 영역을 추출하는 과정을 거치는데 여기서는 문자가 있는 영역을 미리 처리해서 보내기 때문에 카카오 API 의 자체 전처리 과정에서 올바른 처리가 이루어지지 않아 발생하는 것으로 보인다.

B. tesseract_OCR.py

구글에서 제공하는 OCR 오픈 소스이다. 많은 부분에서 사용되지만 한글 인식률은 매우 떨어지는 모습을 보인다. 그러나 영문 OCR 에 있어서는 카카오 API 보다 속도 측면에서 우세하다.

C. Easy_OCR.py

다양한 언어를 지원하고 한글 인식률도 뛰어난 편이다. 다만 CPU 를 사용하여 OCR 을 하였을 때 속도 측면에서 느리다는 단점이 있다. 그러나 GPU 를 사용할 수 있어 속도 측면에서도 개선시킬 수 있는 방안이 보인다. [그림 14]에서 차량의 번호판을 정확히 인식하는 모습을 보여주고 있다.

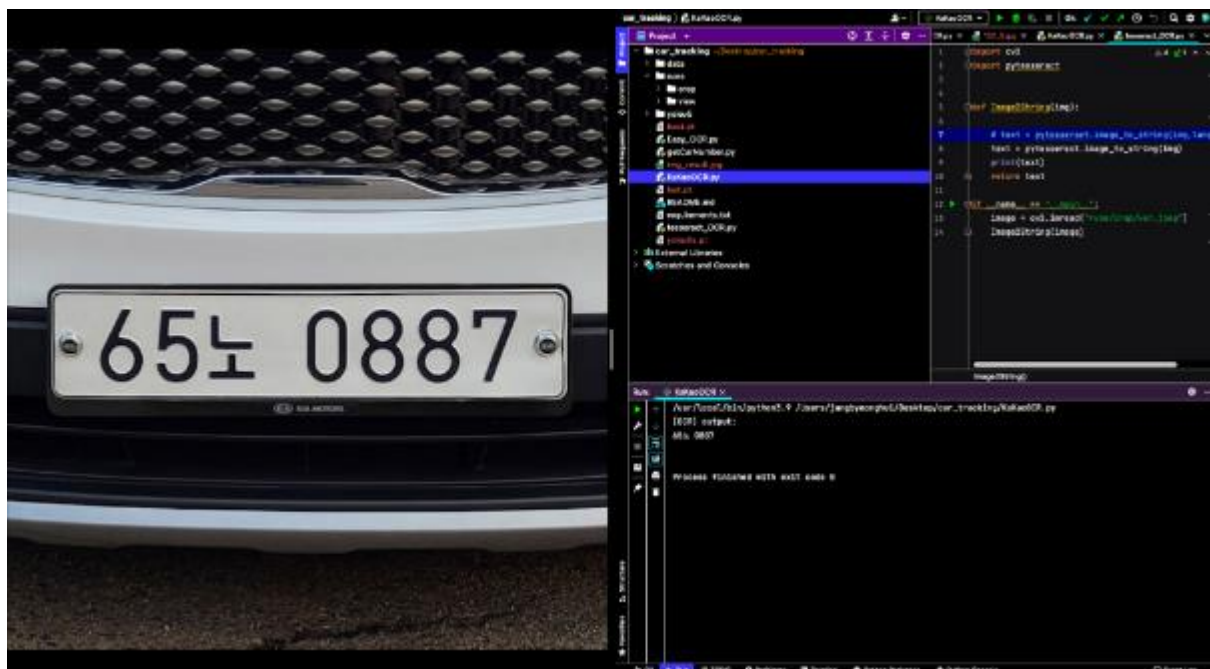


그림 14. EasyOCR 실행 결과

4. YOLO 학습

YOLO 를 통해서 차량에 부착된 번호판을 인식하기 위해서 Kagg1e 에 있는 데이터와 구글에 있는 친환경 자동차의 번호판 이미지를 수집하여 이를 라벨링하였다. 이후 라벨링된 데이터를 바탕으로 학습을 시켜 차량 내에 번호판을 인식할 수 있도록 하였다. 이를 통해서 차량에서 번호판 이미지만을 추출할 수 있게 하였다.

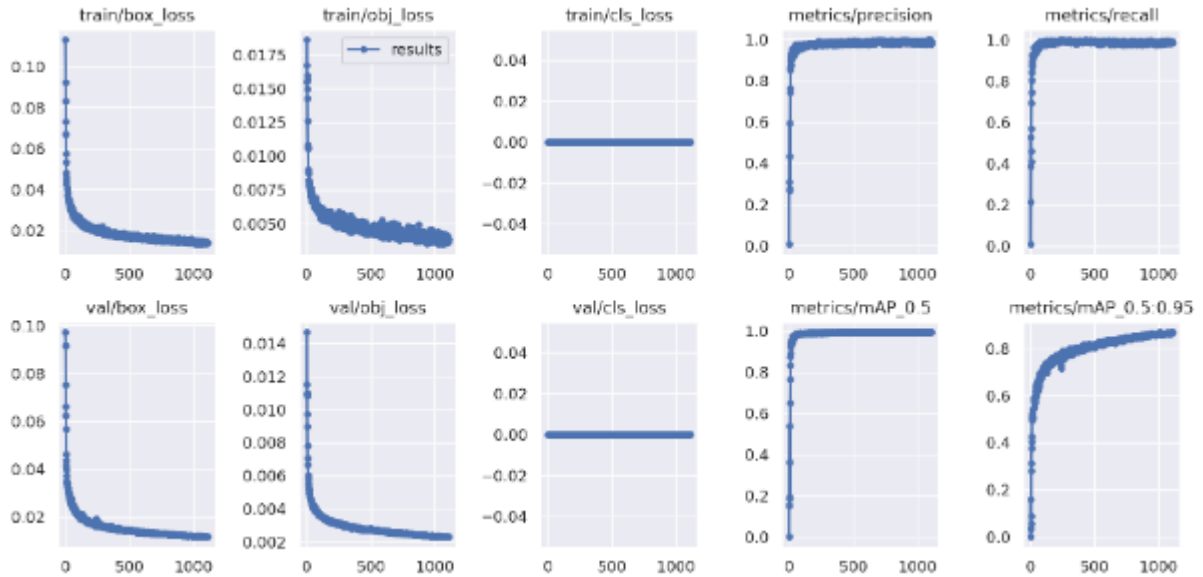


그림 15. YOLO 학습 결과



그림 16. 학습된 가중치로 번호판 인식 결과

5. HSV Masking

[그림 17]의 HSV Masking 툴을 통해서 친환경 자동차 번호판만 검출할 수 있도록 하였고 검출한 색상을 Gaussian Blur 로 변환한 후 픽셀 수를 계산하여 특정 픽셀 이상인 경우 친환경 자동차라고 판단하는 모듈을 개발하였다.



그림 17. HSV Masking 을 통한 친환경 색상 검출

[그림 17]를 통해서 볼 수 있듯이 HSV Masking 을 통해서 번호판들 중에서 친환경 색상 번호판의 색상만 추출하는 것을 확인할 수 있다.



그림 18. 친환경 번호판 추출 결과



그림 19. 친환경 번호판 이진화 결과

[그림 19]에서 볼 수 있듯이 친환경 번호판에 대해서 색상을 검출하고 그 검출한 결과를 이진화하여 dilate 시킨 후 0 이 아닌 픽셀 수를 계산한다. 만약 친환경 차량이라면 픽셀이 검출될 것이며, 친환경 차량이 아니라면 검출되지 않는다. 이를 통해서 친환경 차량인지 아닌지를 구분할 수 있게 된다.

6. 실행 결과

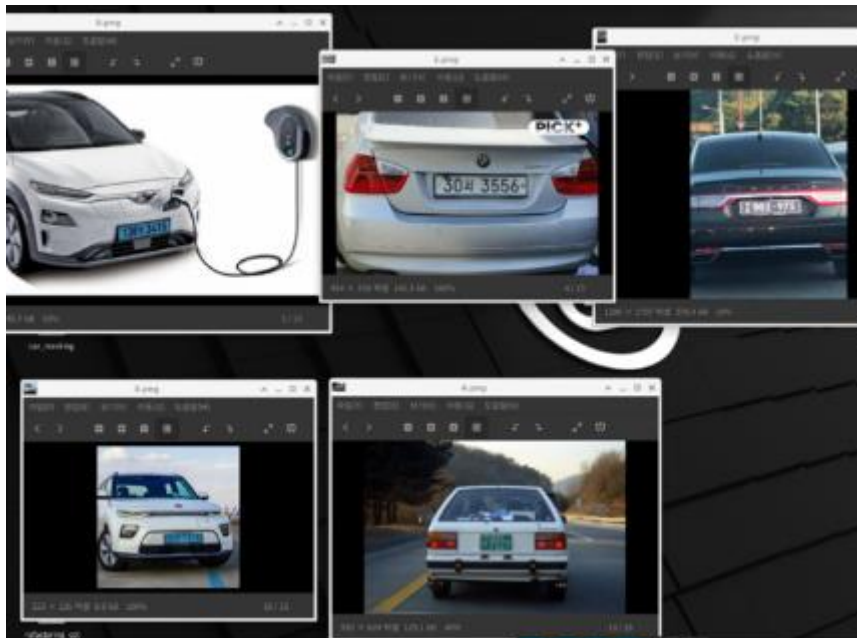


그림 20. 테스트에 사용된 차량 이미지

```
(easyocr) ves@ves-GPU:~/바탕화면/ANPR$ python main.py
Downloading: "https://github.com/ultralytics/yolov5/archive/master.zip" to /home/ves/.cache/torch/hub/master.zip
YOLOv5 2022-5-13 torch 1.8.2 CUDA:0 (GeForce RTX 3090, 24268MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
Adding AutoShape...
친환경 자동차 OCR : 138차 3415
일반 자동차 OCR : 30서 35569
일반 자동차 OCR : 화 001-975
친환경 자동차 OCR : 33부 73145
일반 자동차 OCR : 0서:393나 7327
```

그림 21. OCR 실행 결과

[그림 20]에 표시된 차량 이미지들을 입력으로 얻어진 결과는 [그림 21]에 나와 있다. 결과를 보게 되면 친환경 차량 번호판을 정확히 구분함을 볼 수 있다. 반면 OCR 결과는 약간 부정확한 면이 보이는데 이는 OCR 만을 따로 학습시킴으로써 극복할 수 있는 부분으로 보인다.

[그림 22]에서와 같이 영상에서도 충분히 번호판 검출이 이루어질 수 있음을 보여준다. 이는 번호판 검출을 딥러닝 모델인 Yolo 를 통해서 하였기에 가능한 것으로 보인다. 그러나 실내에 설치된 카메라는 셔터 속도가 느리게 설정되어 있어서 문자가 번진 상태로 보이게 된다. 이는 카메라 해상도와 상관이 없는 영역이다. 하지만 번호판이 잘 보이기만 하면 일반 방법 카메라를 통해서도 문제없이 ALPR 을 할 수 있음을 확인하였다.



그림 22. 영상에서의 번호판 검출 결과

부록

1. 영상분석 서버 내 운영체제 설치 및 환경설정

A. Proxmox 가상화 하이퍼바이저 OS 설치

Proxmox 는 가상화 관리를 위한 소프트웨어 서버로, 호스트+게스트 시스템으로 구성되어 게스트 시스템에 Windows 나 Linux 등의 운영체제를 설치하여 하나의 PC 에 여러 개의 운영체제를 구동할 수 있는 소프트웨어이다. 영상분석 서버에는 다수의 게스트 시스템을 설치할 필요가 없지만, Proxmox 를 사용하여 게스트 시스템에 Linux 운영체제를 설치하여 영상분석 서비스를 가동하면 게스트 시스템에 문제가 발생해도 쉽게 복구할 수 있는 장점이 있다. 특히, 영상분석 서버는 원격지 현장에 설치되어 있는 경우가 많아 서버를 물리적으로 종료하거나 재시작하기 어려운데, 가상화 시스템을 사용하면 호스트 시스템에 오류가 발생하지 않는 한 소프트웨어적으로 게스트 시스템을 제어할 수 있다. 이 단락에서는 영상분석 서버에 Proxmox 호스트 시스템을 설치하고, 게스트 시스템에 Linux 운영체제를 설치하는 방법을 설명한다.

a. Proxmox OS 설치 이미지 다운로드

Proxmox OS 설치 이미지는 Proxmox 공식 홈페이지에서 다운로드 받을 수 있다. 다운로드한 파일은 BalenaEtcher 등의 툴로 USB 드라이브에 쓸 수 있다.

b. 호스트 시스템 설치

컴퓨터의 BIOS (UEFI) 설정에서 Proxmox 설치 파일이 담긴 USB 로 부팅하도록 설정하고 재부팅하면 Proxmox 설치 화면이 나타난다. 다음과 같은 과정으로 설치한다.

i) Install Proxmox VE 항목 선택 [그림 23]



그림 23. Proxmox 설치 화면

ii) 사용 계약에 'I agree' 버튼 클릭 [그림 24]



그림 24. 사용권 계약 동의

- iii) 설치할 저장소 선택. 디스크가 하나만 있으면 상관없으나, 여러 개의 디스크가 설치되어 있을 시 유의 [그림 25]

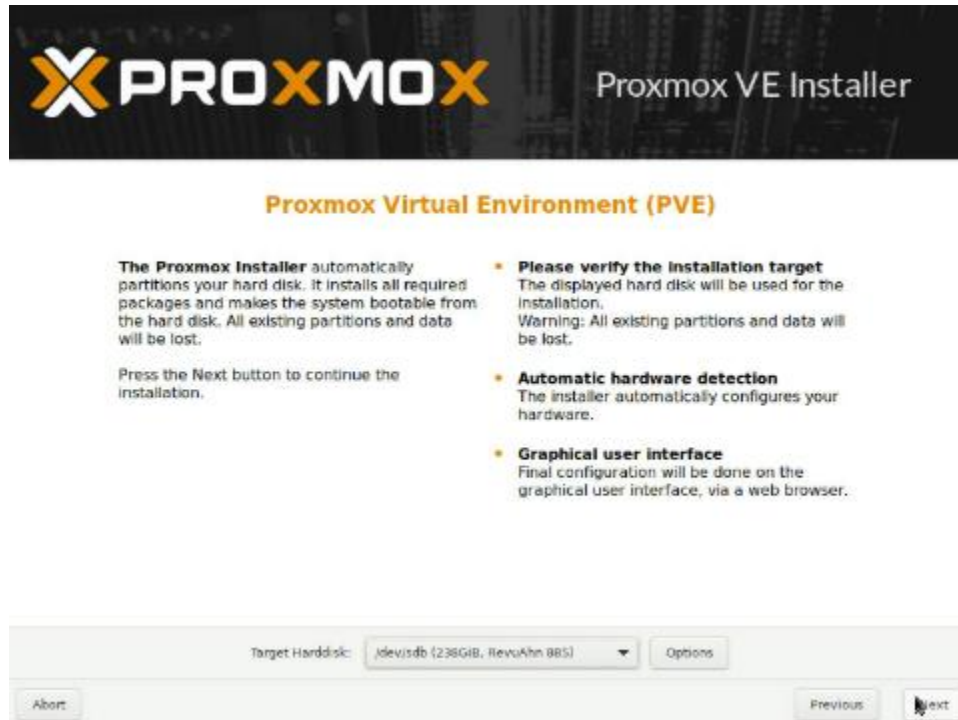


그림 25. Proxmox 설치 디스크 선택

- iv) 시간대 설정. Time zone 은 'Asia/Seoul'로 선택 [그림 26]

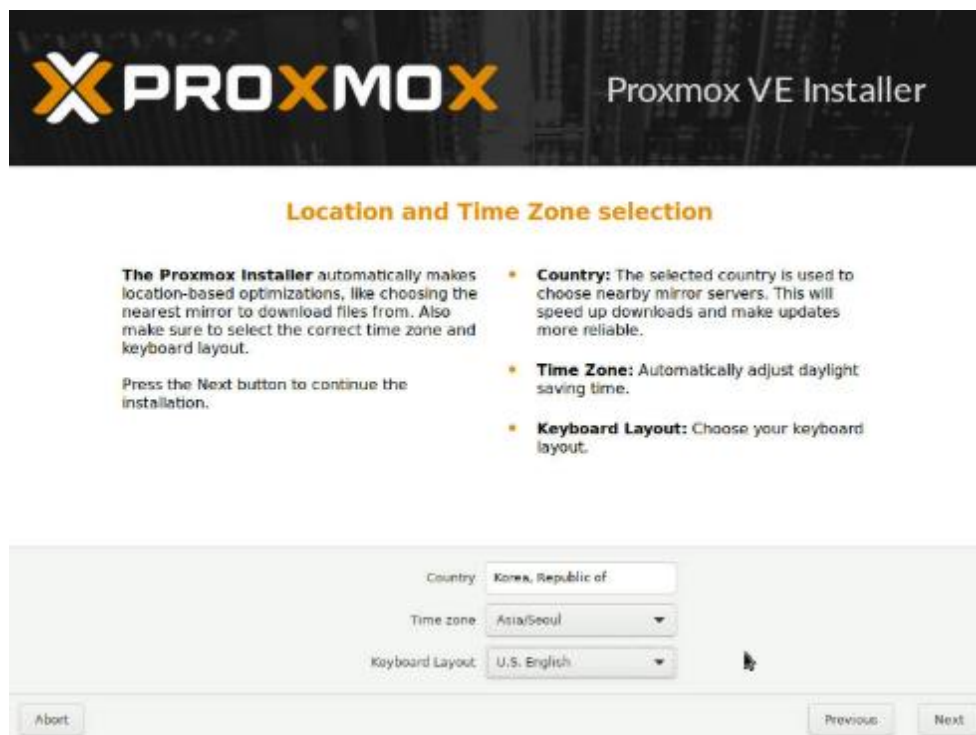
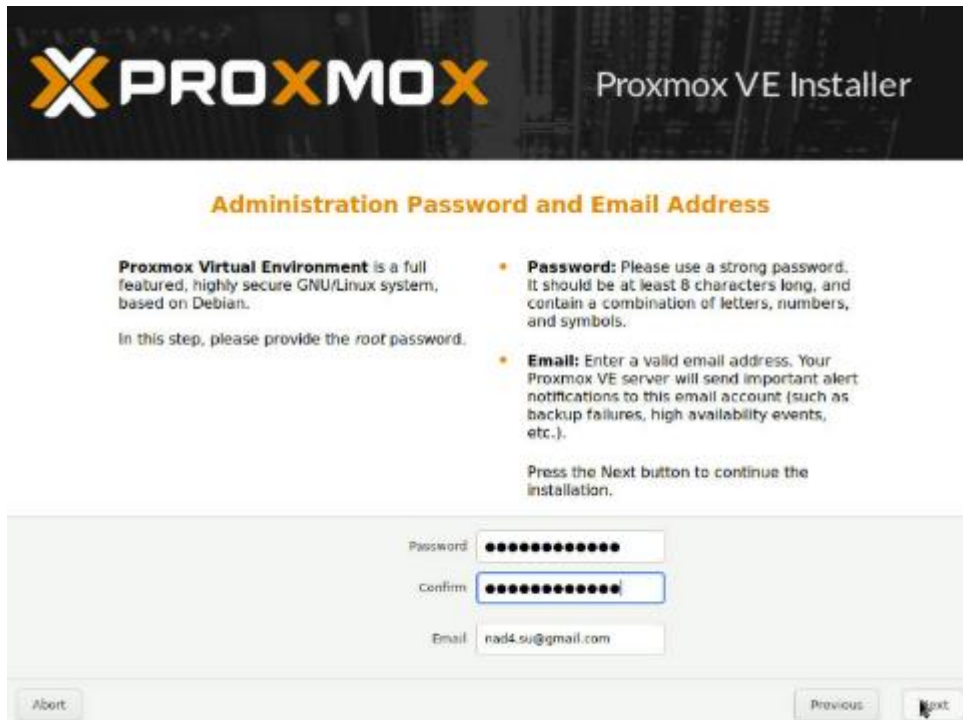


그림 26. 지역 및 시간대 설정

v) 루트 계정의 비밀번호 설정. 암호는 'ves4982!@' 로 입력 [그림 27]



The screenshot shows the 'Administration Password and Email Address' screen of the Proxmox VE Installer. It includes instructions for setting a strong password and a valid email address. The form fields are filled with masked characters for the password and confirm fields, and the email address 'nad4.su@gmail.com' is entered in the email field. Navigation buttons 'About', 'Previous', and 'Next' are visible at the bottom.

PROXMOX Proxmox VE Installer

Administration Password and Email Address

Proxmox Virtual Environment is a full featured, highly secure GNU/Linux system, based on Debian.

In this step, please provide the root password.

- Password:** Please use a strong password. It should be at least 8 characters long, and contain a combination of letters, numbers, and symbols.
- Email:** Enter a valid email address. Your Proxmox VE server will send important alert notifications to this email account (such as backup failures, high availability events, etc.).

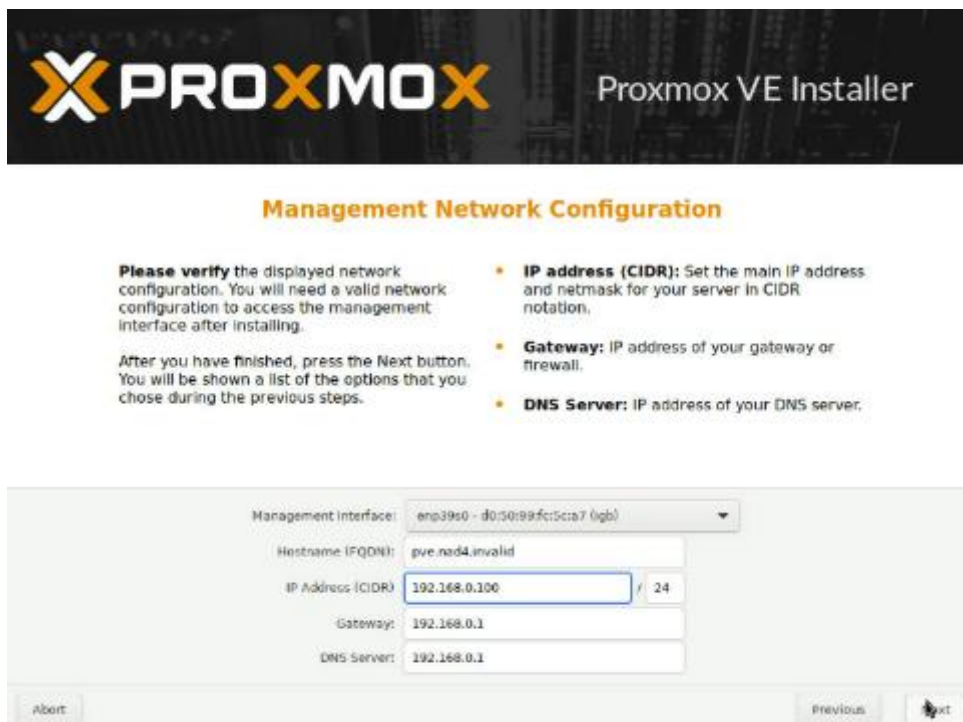
Press the Next button to continue the installation.

Password: [masked]
Confirm: [masked]
Email: nad4.su@gmail.com

Buttons: About, Previous, Next

그림 27. 루트 비밀번호 설정

vi) 네트워크 설정. Hostname 은 'X.prox' (X 는 주차장 약어), IP 및 Gateway, DNS 주소는 현장의 여건에 맞게 설정 [그림 28]



The screenshot shows the 'Management Network Configuration' screen of the Proxmox VE Installer. It includes instructions for verifying the network configuration and setting the IP address, gateway, and DNS server. The form fields are filled with the following values: Management Interface (enp3s0), Hostname (pve.nad4.invalid), IP Address (192.168.0.100), Gateway (192.168.0.1), and DNS Server (192.168.0.1). Navigation buttons 'About', 'Previous', and 'Next' are visible at the bottom.

PROXMOX Proxmox VE Installer

Management Network Configuration

Please verify the displayed network configuration. You will need a valid network configuration to access the management interface after installing.

After you have finished, press the Next button. You will be shown a list of the options that you chose during the previous steps.

- IP address (CIDR):** Set the main IP address and netmask for your server in CIDR notation.
- Gateway:** IP address of your gateway or firewall.
- DNS Server:** IP address of your DNS server.

Management Interface: enp3s0 - d0:50:99:fc:5ca7 (igb)
Hostname (FQDN): pve.nad4.invalid
IP Address (CIDR): 192.168.0.100 / 24
Gateway: 192.168.0.1
DNS Server: 192.168.0.1

Buttons: About, Previous, Next

그림 28. 네트워크 IP 주소 설정

vii) 모든 설정을 다시 확인한 후 설치 진행 [그림 29]

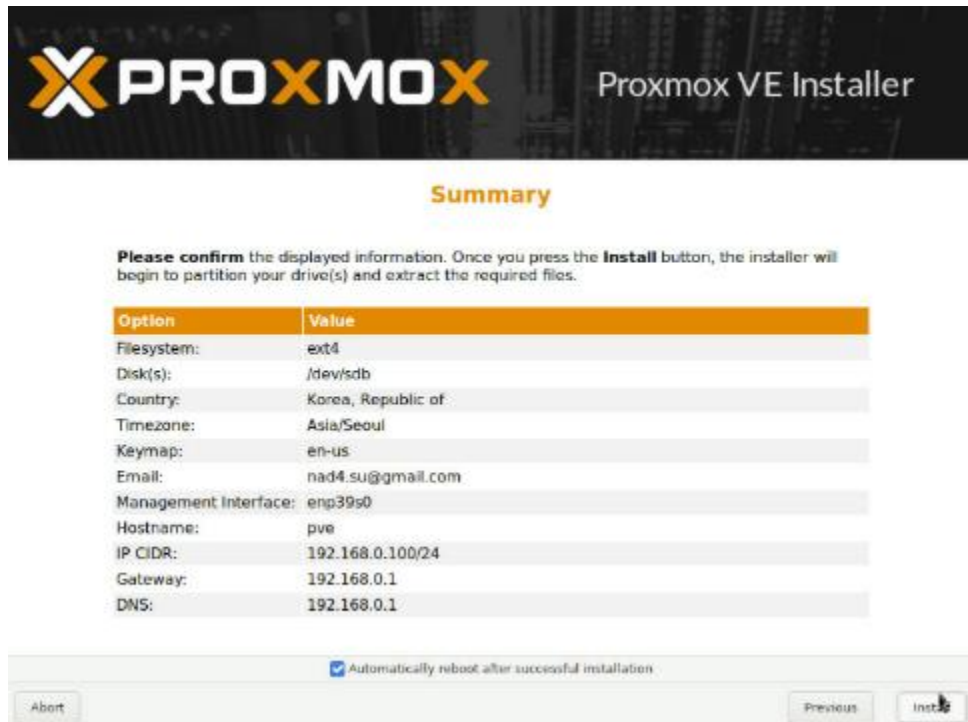


그림 29. 설치 옵션 재확인

viii) 설치가 완료되면 다음 화면이 나타나며 여기에 표시된 IP 주소로 다른 컴퓨터에서 접속 [그림 30]

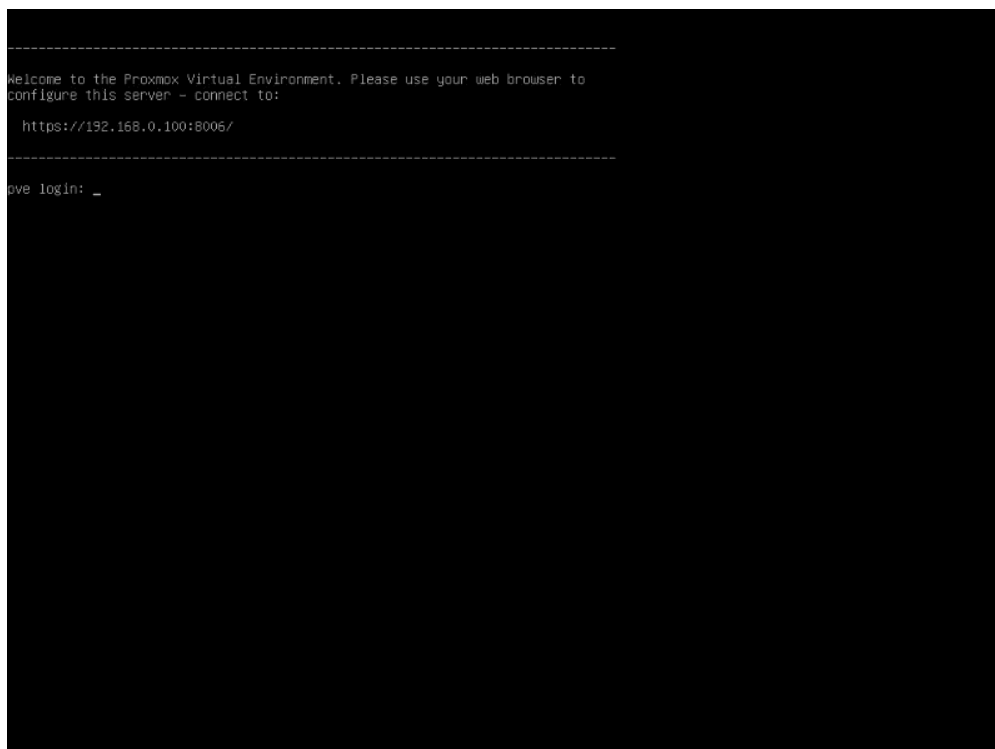


그림 30. 설치 완료 화면

c. 게스트 시스템 설치

호스트 시스템 설치가 완료되면, [그림 31]과 같은 웹페이지에 접속 가능하다. 게스트 시스템 설치 방법은 크게 두 가지가 있는데, 가상 시스템 구동에 필요한 하드웨어 값을 직접 설정하여 OS를 설치하거나 미리 설정해둔 백업 이미지를 복원하여 설치하는 방법이 있다. 영상분석 서버의 경우 사용하는 OS가 리눅스로 동일하기 때문에 후자의 방법으로 설치하는 것이 간단하다. 따라서 이 항목에서는 후자의 방법으로 설치하는 방법을 설명한다.

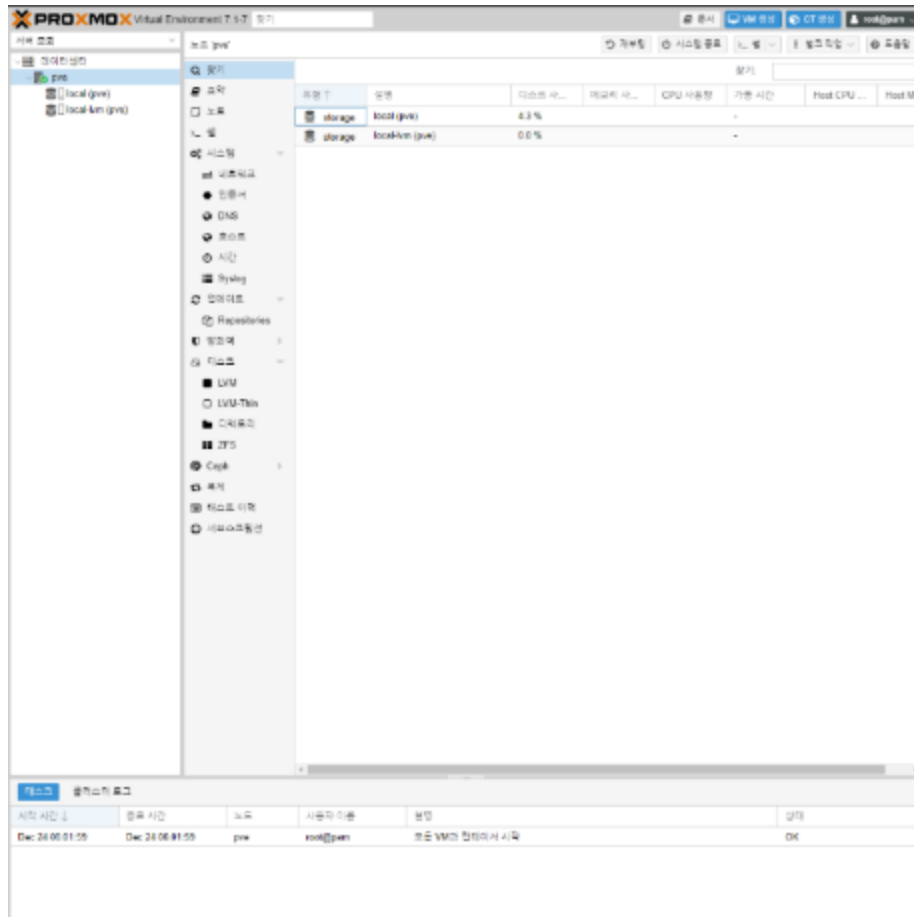


그림 31. Proxmox 호스트 웹페이지

- i) 백업 이미지는 기 설치된 영상분석 서버에서 다운로드 받을 수 있다. [그림 32]와 같이 호스트 시스템의 웹페이지에서 'Shell'을 클릭하여 쉘 스크립트 창을 실행한 후, `scp -P {ssh_port_number} root@{server_ip}:/var/lib/vz/dump/{backup_filename} /var/lib/vz/dump/`를 입력한다. 여기서 `{ssh_port_number}`는 백업 이미지가 저장된 영상분석 서버의 외부 SSH 포트 번호, `{server_ip}`는 영상분석 서버의 IP 주소, `{backup_filename}`은 백업 이미지 파일 이름이다.



그림 32. Proxmox 호스트 셸 스크립트 창

- ii) 백업 이미지 파일 다운로드가 완료되면 [그림 33]과 같이 'local' 드라이브의 'Backups' 항목에 백업 파일이 생성되며, 파일 선택 후 'Restore' 버튼을 클릭하여 [그림 34]과 같이 기존 설정대로 게스트 시스템을 설치할 수 있다.

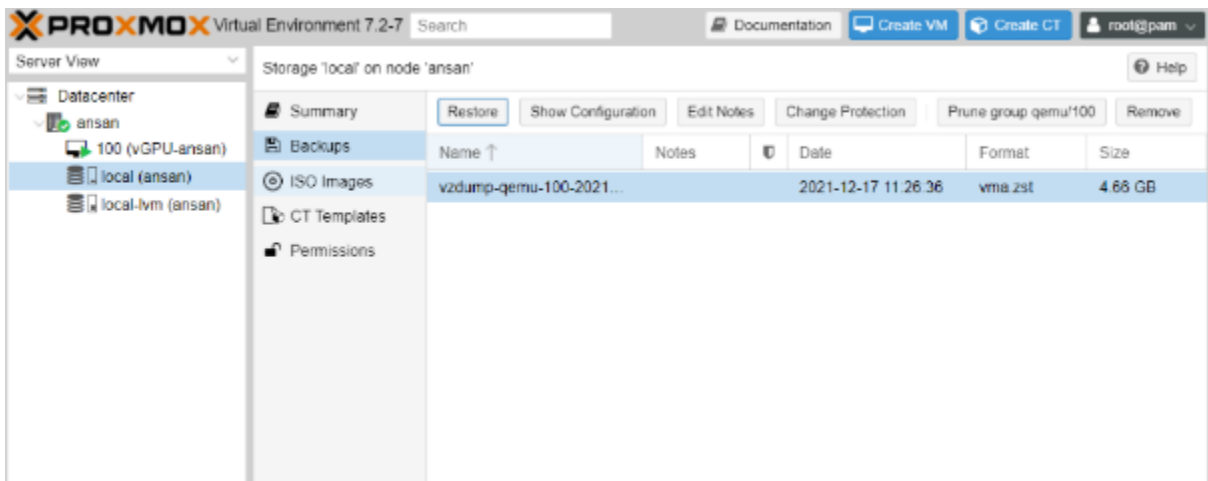


그림 33. 백업 이미지 파일 다운로드

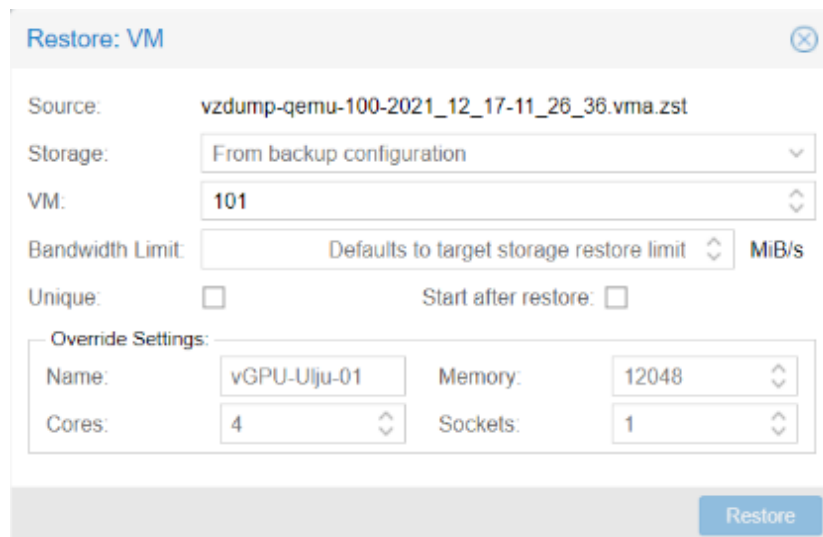


그림 34. 백업 이미지에서 게스트 시스템 복원

! 참고

[그림 34]에서 게스트 시스템 이름, 메모리 크기, CPU 코어 수, CPU 개수를 지정할 수 있다. 서버마다 하드웨어 사양이 상이하므로 호스트 시스템의 하드웨어 리소스를 초과하지 않는 범위 내에서 설정한다.

B. GPU Passthrough 활성화

가상 환경에서 GPU를 사용하기 위해서는 PCI Passthrough 과정이 필요하다. 가상 환경에서 사용하는 하드웨어는 모두 가상화 되어있어 Passthrough를 하지 않으면 PCI 장치의 모델명이 호스트 시스템과 다르게 나타나며, CUDA를 이용한 GPU 가속화 연산 기능을 사용할 수 없다. Passthrough를 활성화하려면 컴퓨터의 BIOS(UEFI) 설정에 진입하여 Virtualization Technology for Directed I/O (VT-d) 또는 SR-IOV (인텔 CPU 메인보드 기준) 또는 IOMMU (AMD CPU 메인보드 기준) 옵션을 활성화해야 한다. Passthrough가 활성화되면 다음 과정을 통해 Passthrough를 수행한다.

- i) Proxmox 호스트 웹페이지에서 생성된 가상 머신으로 진입하여 'Hardware' 탭에 들어간 다음 'PCI Device' 항목의 장치 ID를 수정해야 한다. [그림 35]
- ii) 'Edit' 버튼을 클릭하고 'Device' 목록을 클릭하면 PCI 장치 ID와 이름이 나타나는데, 여기서 GPU 장치를 찾아 클릭한다. [그림 36]
- iii) 이후 'Advanced' 항목에 체크하고 'ROM-Bar' 항목과 'PCI-Express' 항목을 체크한 후 'OK' 버튼을 클릭하여 설정을 저장한다. 그러면 가상 머신에서도 호스트 시스템과 동일한 GPU를 사용할 수 있다. [그림 37]
- iv) GPU Passthrough가 정상적으로 되었는지 확인하려면 가상 머신의 터미널 창에서 '`nvidia-smi`' 명령어를 입력하고 NVIDIA GPU 장치 목록이 출력되는지 확인하면 된다. [그림 38]

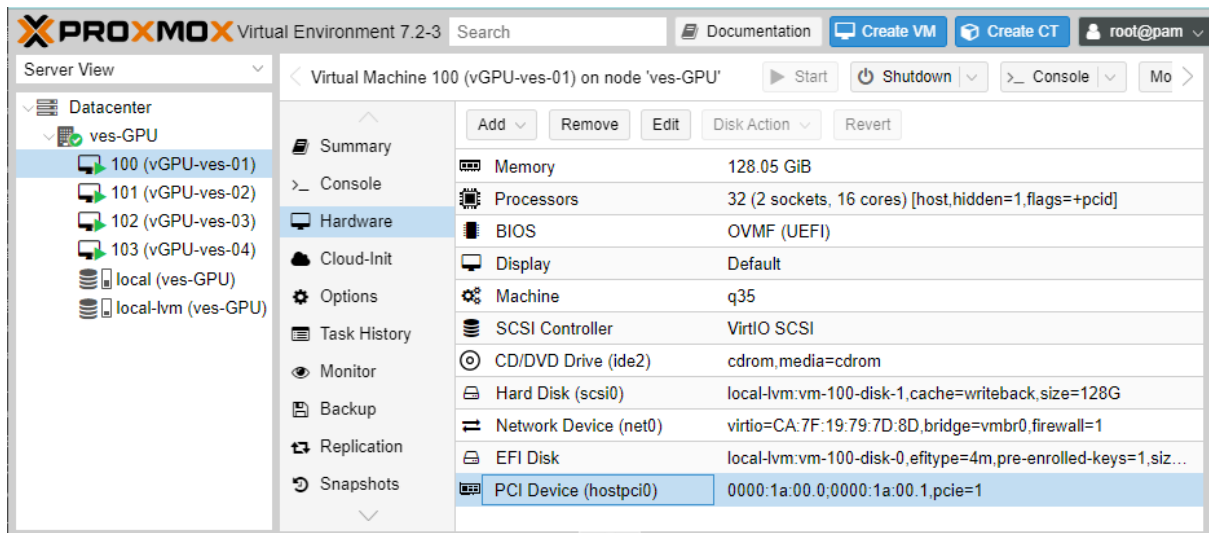


그림 35. 게스트 시스템의 장치 목록

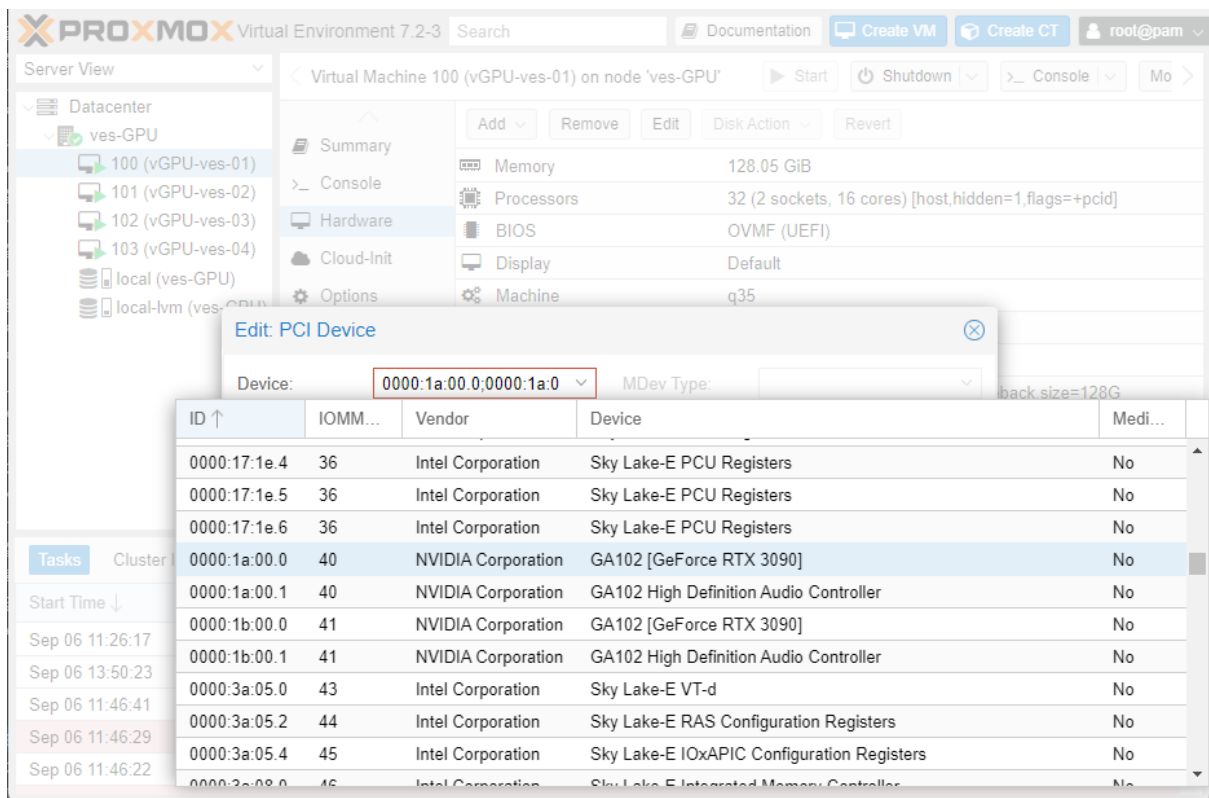


그림 36. 게스트 시스템의 GPU 장치 선택

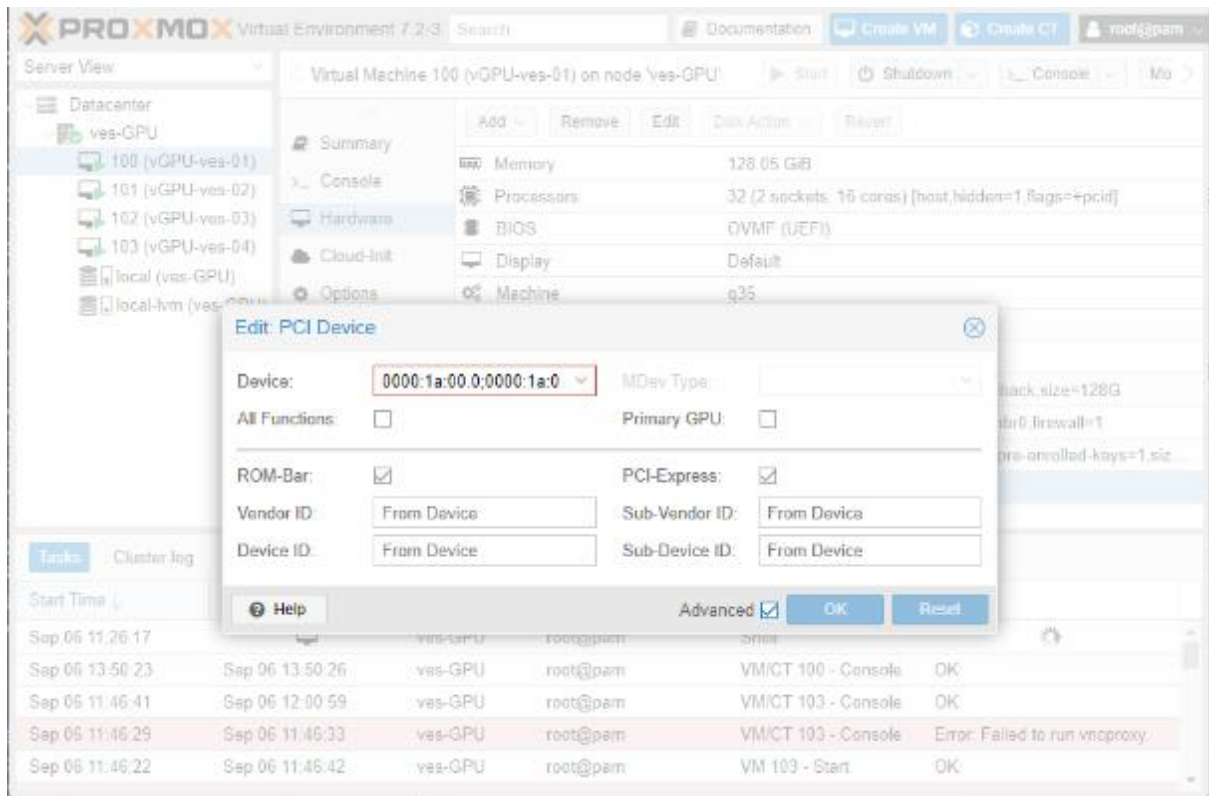


그림 37. GPU 장치 속성 선택

```
(base) ves@vGPU-ves-01:~$ nvidia-smi
Tue Sep  6 14:01:50 2022
+-----+
| NVIDIA-SMI 470.141.03   Driver Version: 470.141.03   CUDA Version: 11.4   |
+-----+-----+-----+-----+-----+-----+
| GPU   Name               Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+
|  0  NVIDIA GeForce ...   Off      | 00000000:01:00.0 Off |           0%        N/A |
| 30%   23C   P8      16W / 350W | 5MiB / 24268MiB |             Default   |
|                               |                      | N/A              |
+-----+-----+-----+-----+-----+-----+
+-----+
| Processes: |
| GPU   GI    CI          PID    Type   Process name                  GPU Memory |
|=====+=====+=====+=====+=====+=====+
|  0   N/A   N/A         1175     G   /usr/lib/xorg/Xorg             4MiB      |
+-----+-----+-----+-----+-----+-----+
+-----+
```

그림 38. GPU 장치 인식 확인

C. Anaconda 가상환경 설치 (Optional)

Anaconda 는 패키지 관리를 용이하게 하기 위한 관리 프로그램이다. 예를 들어, 서버에서 차량 인식 외 다른 프로젝트 (예. 보행자 인식 등)를 실행하려면 필요한 패키지가 다소 상이할 수 있는데, Anaconda 를 사용하면 Anaconda 의 가상 환경별로 패키지를 다르게 설치할 수 있다. 영상분석 서버에서는 차량 인식만 주로 수행하기

때문에 Anaconda 를 굳이 설치하지 않아도 상관없으나, 추후 다른 프로젝트를 실행할 것을 대비해 설치하는 것을 권장한다. 이 단락에서는 Anaconda 를 설치하고 그 내부에 가상 환경을 설정하는 방법을 설명한다.

a. Anaconda 설치 파일 다운 및 실행

Anaconda 버전 목록은 <https://repo.anaconda.com/archive/>에서 확인할 수 있다. 가급적 2021 년 이후 버전을 선택하는 것이 좋으며, 여기서 리눅스 64 비트 버전 (x86_64)의 설치 파일을 다운받으면 된다. 파일 다운로드는 wget 을 이용하여 파일 링크 주소를 복사한 다음 터미널에 다음과 붙여넣고 실행하면 된다.

```
>> wget https://repo.anaconda.com/archive/Anaconda3-*-Linux-x86_64.sh
```

다운받은 파일을 실행하려면 터미널에 `'bash Anaconda3-*-Linux-x86_64.sh'`를 입력한다. 여기서 *은 Anaconda 의 버전명이다. 라이선스 계약에 동의하고 설치한 후 터미널을 다시 실행하면 다음과 같이 계정 이름 앞에 가상환경 이름이 붙는다.

```
(base) ves@vGPU-ves-01:~$
```

b. 가상 환경 생성

base 는 Anaconda 의 기본 가상환경 이름이다. 여기에는 Python 구동을 위한 기본 패키지만 설치되어 있고 다른 패키지는 설치할 수 없다. 따라서 다른 가상환경을 생성하여 영상분석에 사용할 패키지를 설치해야 한다. 다음 과정으로 가상환경을 생성할 수 있다.

- i) 터미널에 `'conda create -n {env_name} python=3.8'`을 입력한다. 여기서 `{env_name}`은 신규로 생성하는 가상환경 이름이고 `{env_name}` 뒤에서 Python 버전을 3.8 로 지정했다. 이를 입력하지 않으면 3.9 버전의 Python 이 설치되는데, Python 3.9 버전에서는 영상분석에 사용되는 일부 패키지가 호환되지 않아 일부러 3.8 버전의 Python 을 설치한다.
- ii) 설치가 완료된 후 터미널에 `'conda activate {env_name}'`을 입력하면 계정 이름 앞의 가상환경 이름이 base 에서 `{env_name}`으로 바뀐다.

D. NVIDIA CUDA Toolkit, CUDNN 설치

GPU 를 이용한 기계학습 연산 가속화 구현을 위해 CUDA Toolkit 및 CUDNN 패키지를 설치한다. CUDA Toolkit 및 CUDNN 설치 방법은 NVIDIA 공식 홈페이지 (<https://docs.nvidia.com/cuda/index.html>)에 자세히 설명하고 있다. CUDA Toolkit 및 CUDNN 은 여러 버전이 있는데, 이 단락에서는 CUDA Toolkit 버전 11.4.1 및 CUDNN 버전 8.2.2 를 기준으로 설치하는 방법에 대해 간략히 소개한다.

⚠ 주의

NVIDIA GPU 모델에 따라 설치 가능한 버전이 한정적이다. NVIDIA에서는 이를 CUDA Compute Capability라 부르며, 이 버전에 따라 설치 가능한 CUDA Toolkit 및 CUDNN 버전을 확인할 수 있다. CUDA Compute Capability 버전 확인은 <https://developer.nvidia.com/cuda-gpus>에서 가능하다.

a. Debian 설치 패키지 파일 다운로드

CUDA 설치 전 gcc 패키지를 설치해야 한다. 설치되어 있지 않으면 ‘`sudo apt-get install gcc`’ 명령어로 먼저 설치한다. 설치 후 ‘`wget https://developer.download.nvidia.com/compute/cuda/11.4.1/local_installers/cuda-repo-ubuntu2004-11-4-local_11.4.1-470.57.02-1_amd64.deb`’ 명령어로 Debian 설치 패키지 파일을 다운로드한다.

b. CUDA 공개 GPG 키 추가

CUDA Toolkit은 리눅스의 기본 설치 Repository에 포함되어 있지 않기 때문에 Repository에 등록할 키 파일을 다운받아 설치해야 한다. [표 33]의 명령어로 키 파일을 설치한 후, 다운받았던 Debian 설치 패키지 파일을 설치한다.

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
$ sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ sudo dpkg -i cuda-repo-ubuntu2004-11-4-local_11.4.1-470.57.02-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/7fa2af80.pub
```

표 33. CUDA Public GPG Key 설치

c. CUDA 설치

‘`sudo apt-get update`’로 APT Repository를 업데이트한 후, ‘`sudo apt-get install cuda`’로 CUDA Toolkit을 설치한다.

d. 시스템 환경변수 추가

시스템에서 CUDA Toolkit 버전을 인식할 수 있도록 시스템 환경변수에 추가하는 작업이 필요하다. 먼저 ‘`export PATH=/usr/local/cuda-11.4/bin${PATH:+:${PATH}}`’ 명령어를 실행하여 CUDA 패키지 경로를 환경변수에 추가하고, ‘`export LD_LIBRARY_PATH=/usr/local/cuda-11.4/lib64\${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}`’ 명령어를 실행하여 라이브러리 목록에 추가한다.

e. CUDNN 설치

NVIDIA 개발자 프로그램 홈페이지(<https://developer.nvidia.com/accelerated-computing-developer>)에서 tar 형식의 CUDNN 설치 파일을 다운로드한다. (NVIDIA 회원가입 필요) 다운로드 후 'tar -xvf cudnn-linux-x86_64-8.2.2.26_cuda11.4-archive.tar.xz' 명령어로 tar 파일의 압축을 해제한 후, [표 34]의 명령어로 CUDA Toolkit 디렉토리에 해당 파일을 복사하면 된다.

```
$ sudo cp cudnn-*-archive/include/cudnn*.h /usr/local/cuda/include
$ sudo cp -P cudnn-*-archive/lib/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn*.h
/usr/local/cuda/lib64/libcudnn*
```

표 34. CUDNN 파일 복사

f. 설치 확인

[표 35]의 명령어로 CUDA Toolkit 및 CUDNN 버전을 확인할 수 있다.

```
$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Wed_Jul_14_19:41:19_PDT_2021
Cuda compilation tools, release 11.4, V11.4.100
Build cuda_11.4.r11.4/compiler.30188945_0
$ cat /usr/local/cuda/include/cudnn_version.h | grep CUDNN_MAJOR -A 2
#define CUDNN_MAJOR 8
#define CUDNN_MINOR 2
#define CUDNN_PATCHLEVEL 2
--
#define CUDNN_VERSION (CUDNN_MAJOR * 1000 + CUDNN_MINOR * 100 +
CUDNN_PATCHLEVEL)
```

표 35. CUDA Toolkit, CUDNN 버전 확인 및 결과

E. OpenCV 설치

영상을 읽고 쓰기 위한 라이브러리로, 핵심 기능은 C/C++로 개발되었지만 Python으로 바인딩되어 있어 Python용 OpenCV 패키지를 설치하면 Python에서 OpenCV를 사용할 수 있다. 설치 방법은 크게 두 가지가 있는데, pip에서 Python용 OpenCV 패키지를 설치하거나 특정 옵션을 추가하여 gcc로 OpenCV 라이브러리를 빌드한 후 설치하는 방법이 있다. 일반적으로 전자의 방법으로 설치하는 것이 간단하나, 특정 비디오를 읽는데 문제가 발생하면 후자의 방법으로 설치해야 영상분석이 가능할 수도 있으니 이 단락에서는 두 가지 방법 모두를 설명한다.

a. pip를 이용한 설치

터미널에 'pip3 install opencv-python'을 입력하면 설치가 완료된다.

b. 사용자 지정 옵션을 이용한 설치

특정 옵션을 추가하여 OpenCV 를 설치하고자 할 때 사용한다. 구체적인 설치 방법은 다음 과정과 같이 정리했다.

- i) OpenCV 구동을 위한 사전 패키지를 [표 36]와 같이 설치한다. 각 패키지에 대한 설명은 [표 37]에서 설명하고 있다.

```
>> sudo apt-get install build-essential cmake pkg-config libjpeg-dev
libtiff5-dev libpng-dev libavcodec-dev libavformat-dev libswscale-dev
libxvidcore-dev libx264-dev libxine2-dev libv4l-dev v4l-utils libgtk-3-
dev mesa-utils libgl1-mesa-dri libgtkgl2.0-dev libgtkglext1-dev
libatlas-base-dev gfortran libeigen3-dev python3-dev python-numpy
python3-numpy
```

표 36. OpenCV 용 사전 패키지 설치 명령어

C/C++ 컴파일러 및 관련 라이브러리 패키지	build-essential, cmake
컴파일 설정 및 링커 플래그 추가 도구	pkg-config
이미지 파일 로딩 패키지	libjpeg-dev, libtiff5-dev, libpng-dev
비디오 파일 로딩 패키지	libavcodec-dev, libavformat-dev, libswscale-dev, libxvidcore-dev, libx264-dev, libxine2-dev
실시간 비디오 캡처 지원 패키지	libv4l-dev, v4l-utils
이미지/영상 출력 GUI 창 도구	libgtk-3-dev 또는 libqt4-dev 또는 libqt5-dev
OpenGL 지원 라이브러리	mesa-utils, libgl1-mesa-dri, libgtkgl2.0-dev, libgtkglext1-dev
OpenCV 최적화 라이브러리	libatlas-base-dev gfortran libeigen3-dev
OpenCV-Python 바인딩 패키지	python3-dev, python3-numpy

표 37. 사전 패키지의 역할

- ii) OpenCV, OpenCV-contrib 의 Github 페이지에서 소스코드를 다운로드한 후
컴파일할 build 디렉토리를 생성하고 이동한다. [표 38]

```
>> sudo apt-get install git
>> git clone https://github.com/opencv/opencv
```

```
>> git clone https://github.com/opencv/opencv\_contrib
>> cd opencv
>> mkdir build && cd build
```

표 38. OpenCV, OpenCV-contrib C++용 소스코드 다운로드

iii) cmake 를 이용하여 OpenCV 의 컴파일 옵션을 설정한다. 각 옵션에 대한 설명은

https://docs.opencv.org/4.6.0/db/d05/tutorial_config_reference.html
을 참고하기 바란다. [표 39]

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_GENERATE_PKGCONFIG=ON \
-D OPENCV_ENABLE_NONFREE=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/.opencv_contrib/modules \
-D INSTALL_C_EXAMPLES=ON \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D BUILD_EXAMPLES=ON \
-D BUILD_DOCS=OFF \
-D BUILD_SHARED_LIBS=ON \
-D BUILD_opencv_python2=OFF \
-D BUILD_opencv_python3=ON \
-D BUILD_NEW_PYTHON_SUPPORT=ON \
-D WITH_CUDA=ON \
-D WITH_CUBLAS=ON \
-D WITH_CUDNN=ON \
-D CUDA_FAST_MATH=1 \
-D CUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-11.4 \
-D OPENCV_DNN_CUDA=ON \
-D CUDA_ARCH_BIN=8.6 \
-D CUDA_ARCH_PTX=8.6 \
-D CUDNN_VERSION=8.2 \
-D CUDNN_INCLUDE_DIR=/usr/local/cuda-11.4/include \
-D CUDNN_LIBRARY=/usr/local/cuda-11.4/lib64/libcudnn.so.8.2.2 \
-D WITH_VTK=ON \
-D WITH_OPENCL=ON \
-D OPENCV_SKIP_PYTHON_LOADER=ON \
-D PYTHON_EXECUTABLE=~/.anaconda3/bin/python3 \
-D PYTHON3_INCLUDE_DIR=~/.anaconda3/include/python3.8 \
-D PYTHON3_NUMPY_INCLUDE_DIRS=~/.anaconda3/lib/python3.8/site-packages/numpy/core/include \
-D PYTHON3_PACKAGES_PATH=~/.anaconda3/lib/python3.8/site-packages \
-D PYTHON3_LIBRARY=~/.anaconda3/lib/libpython3.8.so \
-D PYTHON_LIBRARIES=~/.anaconda3/lib/python3.8 ..
```

표 39. cmake 의 컴파일 옵션 지정 명령어

iv) [그림 39]과 같은 메시지가 출력되면 빌드에 성공한 것이다.

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/webnautes/opencv/opencv-4.2.0/build
```

그림 39. 빌드 성공 메시지 출력

- v) `nproc` 명령어로 컴퓨터의 CPU 코어 개수를 확인한 후, `make -j{n}` 명령어로 컴파일을 시작한다. 여기서 `{n}`은 `nproc` 명령어로 확인한 CPU 코어 개수이다.
- vi) '`sudo make install`' 명령어를 입력하여 컴파일된 OpenCV 패키지를 설치한다.
- vii) 설치 완료 후 OpenCV 라이브러리 사용 가능 여부를 [표 40]과 같이 확인

```
>> python3
>> import cv2
>> cv2.__version__
'4.6.0'
```

표 40. OpenCV 정상 설치 여부 확인

F. PyTorch 및 YoloV5 필수 라이브러리 설치

PyTorch는 기계학습 라이브러리 중 하나다. 기계학습 라이브러리는 PyTorch 외에도 TensorFlow, Keras, ONNX, TensorRT 등이 있다. PyTorch는 Define by Run 패러다임을 갖고 있어 이해와 디버깅이 쉽고 학습 및 추론 속도가 빠른 장점을 갖고 있다. 이 단락에서는 PyTorch를 설치하고 YoloV5 구동에 필요한 라이브러리를 설치하는 방법에 대해 설명한다.

⚠ 주의

GPU를 사용할 경우 반드시 PyTorch를 먼저 설치한 후 YoloV5 필수 라이브러리를 설치해야 한다. YoloV5의 라이브러리 설치 파일 (`requirements.txt`)에도 PyTorch를 설치할 수 있으나 CPU 전용 패키지라서 이를 설치하면 GPU 가속을 사용할 수 없다.

a. PyTorch 설치 파일 다운로드 및 설치

GPU를 사용할 경우 CUDA 버전에 맞는 설치 파일을 다운로드한다. 설치 방법은 Anaconda의 conda나 pip를 이용하여 설치할 수 있다. PyTorch는 2022년 9월 현재 버전 1.12.1까지 출시되었으나 YoloV5내 라이브러리와 호환성 문제로 1.10.1 버전을 설치하는 것을 권고한다. [표 41]의 명령어로 pip를 이용하여 PyTorch를 설치할 수 있다.

```
$ pip install torch==1.10.1+cu111 torchvision==0.11.2+cu111  
torchaudio==0.10.1 -f https://download.pytorch.org/whl/torch_stable.html
```

표 41. pip 를 이용한 PyTorch 설치

b. YoloV5 필수 라이브러리 설치

YoloV5 의 프로젝트 루트 디렉토리에 있는 `requirements.txt` 파일에 YoloV5 구현에 필요한 라이브러리 목록이 있다. 이를 ‘`pip install -r requirements.txt`’ 명령어를 실행하여 모두 설치할 수 있다.

2. 영상분석 서버 접속 IP 리스트 및 계정 정보

현재 서비스 중인 영상분석 서버 및 개발용 서버에 원격으로 접속하기 위한 IP 주소 및 접속 프로토콜 별 포트 번호를 [표 42]와 같이 정리했다. 해당 IP 는 사내 네트워크에서만 접속 가능하며, 일부 서버 및 프로토콜은 접속하고자 하는 PC 에 VPN Proxy 를 설치해야만 접속 가능하다.

접속 시 ID 및 비밀번호를 묻는 창이 나타날 것인데, ID 는 ves, 비밀번호는 qwe123,.을 입력하면 된다. 단, Proxmox 호스트 시스템에 로그인할 때는 ID 를 root, 비밀번호는 ves4982!@를 입력해야 한다.

서버 이름	IP 주소	프로토콜 포트 번호			비고
		SSH	RDP	Web (HTTPS)	
안양 2 차 SKV1	115.144.111.254	51060	51061	-	서비스용
안양 2 차 SKV1-Test	115.144.111.254	31002 (Guest)	31003 (Guest)	31001 (Host)	테스트용, Proxmox
안양 2 차 SKV1-NVR	1.255.252.24	-	501	-	모니터링 PC (윈도우 10)
천호역 공영주차장	115.144.111.254	51000 (Host)	-	51001 (Host)	서비스용, Proxmox
	10.2.0.13	41010 (Guest)	41011 (Guest)	-	Proxy Server
	192.168.0.31	51012 (Guest)	51013 (Guest)	-	Raspberry Bridge
강동역 공영주차장	115.144.111.254	51010 (Host)	-	51011 (Host)	서비스용, Proxmox
	10.2.0.13	41020 (Guest)	41021 (Guest)	-	Proxy Server
	192.168.0.31	51014 (Guest)	51015 (Guest)	-	Raspberry Bridge
호반파크 2 관	115.144.111.254	51110 (Guest)	51112 (Guest)	51111 (Host)	서비스용, Proxmox
안산 상하수도	115.144.111.254	51082 (Guest)	51083 (Guest)	51081 (Host)	서비스용, Proxmox
	192.168.0.19	-	-	-	IPMI 웹페이지

개발용 GPU 서버	192.168.0.146	22	-	8006	Proxmox Host
	192.168.0.126	22	-	3389	VM 100 (Guest)
	192.168.0.155	22	3389	-	VM 101 (Guest)
	192.168.0.189	-	3389	-	VM 102 (Guest)
	192.168.0.153	22	3389	-	VM 103 (Guest)

표 42. 영상분석 서버 접속 정보

3. 영상분석용 주차장 CCTV RTSP 접속 정보

영상분석에 사용되는 주차장 내 CCTV의 영상에 접속할 수 있는 주소를 별도의 엑셀 파일에 정리했다. 영상은 RTSP 프로토콜을 이용하여 시청할 수 있으며, 이는 기본적으로 영상분석 서버에 접속한 후 가능하다. 다만 일부 주차장은 VPN Proxy를 설치하면 해당 PC에서도 시청 가능하다. 영상에 접속할 때는 동영상 플레이어의 URL 입력 창에 ‘`rtsp://{ID}:{PW}@{IP}:{Port}/{args}`’ 형식으로 입력한다.

4. 주차장 수동 관제 모드 실행방법

영상분석이 적용되는 주차장은 주차면 점유 판단 정확도가 차량 가림 또는 이중주차 등의 사유로 100%가 되지 않는다. 따라서 수동으로 주차면 판단 결과를 변경하고자 할 때는 주차면 관제 모니터링 웹페이지에서 판단 결과를 변경할 수 있다. 이 기능은 안양 2차 SKV1 및 안산 상하수도사업소 공영주차장에만 제공되고 있으며, 모니터링 웹페이지 주소는 [표 43]에 정리했다.

주차장 이름	웹페이지 주소
안양 2차 SKV1	https://skv1-ay2.watchmile.com/static_slot/
안산 상하수도사업소 공영주차장	http://ansan.watchmile.com:9080/

표 43. 주차면 수동 관제 웹페이지 주소 목록

수동으로 주차면 점유 결과를 변경할 때는 영상분석 프로세스를 종료해야 변경된 결과를 유지할 수 있다. 따라서 [그림 6]의 작업 스케줄러에 등록된 스크립트 실행을 비활성화한 후 (주석 처리), ‘`pkill -9 -ef run_`’ 명령어로 현재 실행중인 영상분석 프로세스를 종료해야 한다. 그리고 웹페이지에 접속한 후 [그림 40]와 같이 주차면을 클릭하면 점유 결과가 변경된다.

⚠ 참고

안산 상하수도사업소 공영주차장의 관제 웹페이지에서는 변경하고자 하는 주차면을 마우스 우클릭하여 자물쇠 아이콘이 뜨면 변경 가능하다. 자물쇠 아이콘이 떠있는 주차면은 수동으로 주차 상태 변경만 가능하다. 다시 마우스 우클릭하면 자물쇠 아이콘이 사라진다.

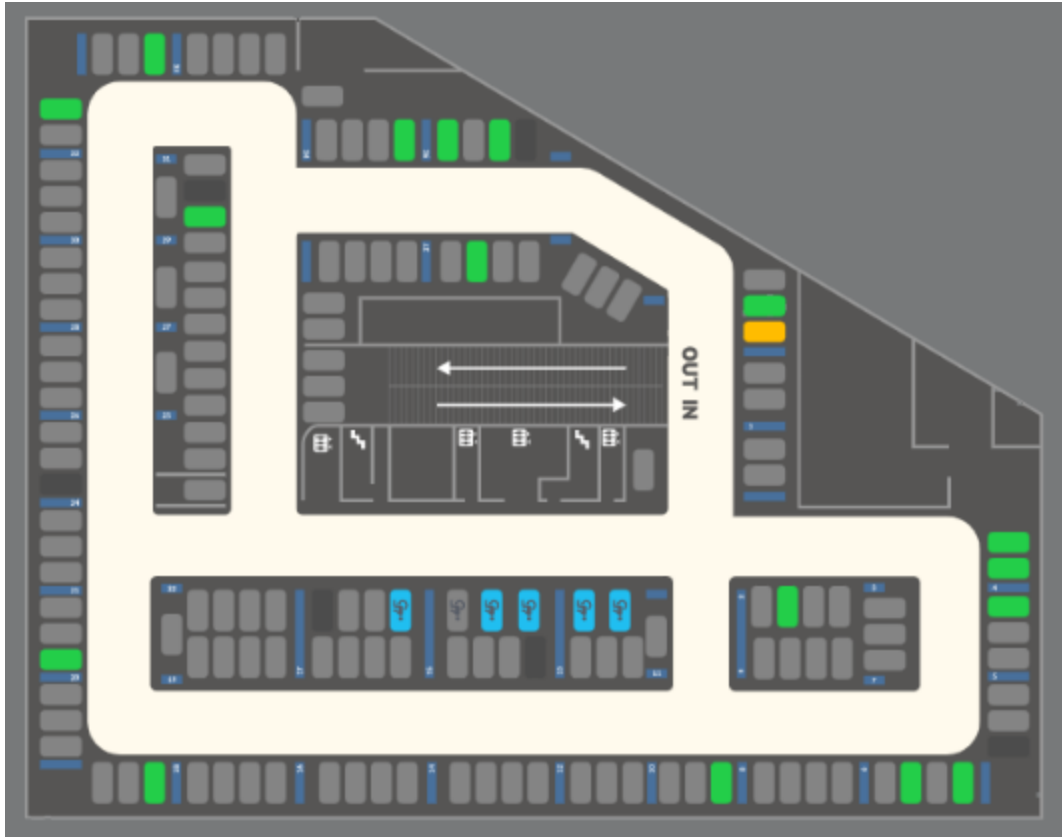


그림 40. 주차면 수동 관제 웹페이지 화면

5. 디스크 파티션 공간 조정 방법 (리눅스 기준)

부록 1.A.c 절과 같이 백업 이미지 파일로 리눅스 게스트 시스템을 설치했을 때 가상 머신 내 파티션 크기는 32GB 로 고정되어 있다. 32GB 로는 영상분석용 패키지 및 소스코드를 설치하기에는 공간이 부족하기 때문에 파티션 용량을 확장해주어야 한다. 이 단락에서는 파티션 용량을 확장하는 방법을 설명한다.

A. Proxmox 호스트 웹페이지에서 디스크 용량 증가

우선 Proxmox 호스트 웹페이지에 접속하여 게스트 시스템의 디스크 용량을 늘려야 한다. [그림 41]와 같이 게스트 시스템의 'Hardware' → 'Hard Disk (scsi0)' 항목을 클릭한 후, 'Disk Action' 버튼을 누르고 'Resize' 버튼을 클릭한다. 그리고 [그림 42]와 같이 추가할 용량을 입력하고 'Resize disk' 버튼을 클릭하면 디스크 용량이 증가한다.

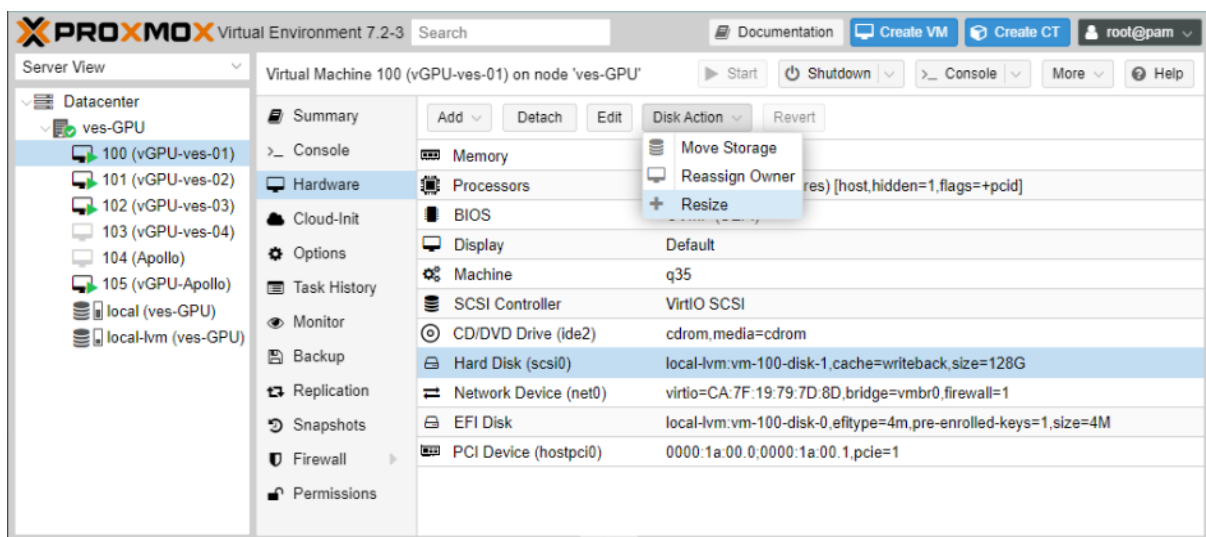


그림 41. Proxmox 호스트 시스템에서 게스트 시스템 디스크 크기 변경

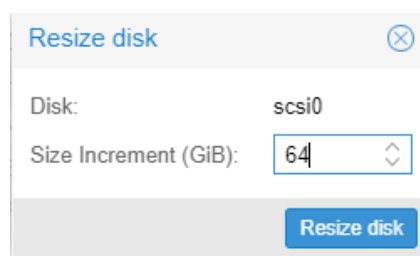


그림 42. 추가할 디스크 공간 입력

B. 게스트 시스템 내 파티션 용량 증가

호스트 시스템에서 디스크 용량을 변경했으면 게스트 시스템 내부에서도 변경된 디스크 용량을 인식하도록 해야 한다. 게스트 시스템에 SSH 프로토콜로 접속한 후, 터미널 명령창에 `df -h` 를 입력하면 [표 44]와 같은 결과를 얻을 수 있다.

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	63G	0	63G	0%	/dev
tmpfs	13G	1.5M	13G	1%	/run
/dev/sda2	126G	59G	62G	49%	/
tmpfs	63G	0	63G	0%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	63G	0	63G	0%	/sys/fs/cgroup
/dev/sda1	511M	5.3M	506M	2%	/boot/efi
tmpfs	13G	8.0K	13G	1%	/run/user/121
tmpfs	13G	4.0K	13G	1%	/run/user/1000

표 44. df -h 를 이용한 파티션 목록 및 용량 확인 결과 예시

영상분석 패키지 및 소스코드가 저장되는 파티션은 ‘/dev/sda2’ 이며, [표 44]에서 보면 디스크 용량이 증가하지 않았음을 확인할 수 있는데, 이를 변경하기 위해서는 growpart 유틸리티로 파티션 공간을 늘려야 한다. growpart 유틸리티는 기본적으로 설치되지 않기 때문에 [표 45]의 명령어로 먼저 설치한 후 resize2fs 로 파티션 공간을 늘릴 수 있다.

```
$ sudo apt-get install cloud-guest-utils
$ sudo growpart /dev/sda 2
$ sudo resize2fs /dev/sda2
```

표 45. growpart, resize2fs 를 이용한 파티션 크기 증가

파티션 공간을 늘린 후 다시 ‘df -h’ 명령어를 입력하면 [표 46]와 같이 파티션 공간이 증가했음을 확인할 수 있다.

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	63G	0	63G	0%	/dev
tmpfs	13G	1.5M	13G	1%	/run
/dev/sda2	254G	59G	62G	24%	/
tmpfs	63G	0	63G	0%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	63G	0	63G	0%	/sys/fs/cgroup
/dev/sda1	511M	5.3M	506M	2%	/boot/efi
tmpfs	13G	8.0K	13G	1%	/run/user/121
tmpfs	13G	4.0K	13G	1%	/run/user/1000

표 46. 증가한 파티션 크기 결과

⚠ 참고

영상분석용 게스트 시스템 파티션의 최소 크기는 64GB 가 적절하다. 실제 디스크 크기가 충분히 크다면 128GB 로 설정해도 되나, 호스트 시스템이 사용할 파티션의 크기도 고려해야 한다.

6. NVIDIA Jetson Nano 설치 및 환경설정

Jetson Nano 는 NVIDIA 에서 개발한 임베디드 컴퓨팅 보드의 한 종류이다. 제품 성능에 따라 TK1, TX1, TX2, Nano 로 구분한다. 라즈베리 파이와 마찬가지로 CPU 는 ARM 아키텍처 프로세서를 사용하지만 Jetson 은 여기에 GPU 를 포함하는 Tegra 칩셋을 탑재하고 있어 GPU 를 이용한 기계학습 모델을 구동할 수 있는 장점이 있다. 그리고 일반 PC 에 비해 전력 사용량이 낮고 부피도 작기 때문에 휴대성이 뛰어나다. 물론 Tegra 칩셋에 탑재된 GPU 는 데스크톱이나 서버에서 사용하는 GPU 에 비해 성능이 극도로 제한되어 있기 때문에 Jetson 을 이용하여 영상분석 서비스를 제공하기에는 한계가 있다. 또한 Jetson 에서 사용하는 OS 는 데스크톱이나 서버에서 사용하는 리눅스 운영체제 (예. Ubuntu, Debian, Mint)를 설치하기에는 사양이 낮아 NVIDIA 에서 Jetson 용 전용 운영체제를 제공하고 있다. 따라서 이 단락에서는 Jetson 용 운영체제 설치 및 영상분석용 라이브러리 설치 방법을 설명한다.

A. JetPack 설치

JetPack 은 Ubuntu Desktop (18.04.5)에 CUDA Toolkit, OpenCV, TensorRT 등의 드라이버가 포함된 Jetson 전용 운영체제이다. [그림 43]과 같이 JetPack 이미지 파일을 다운로드한 후, [그림 44]과 같이 Win32 Disk Manager 를 이용해 이미지 파일을 SD 카드에 기록한다. 이후 SD 카드를 Jetson Nano 에 설치하고 전원을 연결하면 Jetson Nano 가 자동으로 부팅된다.

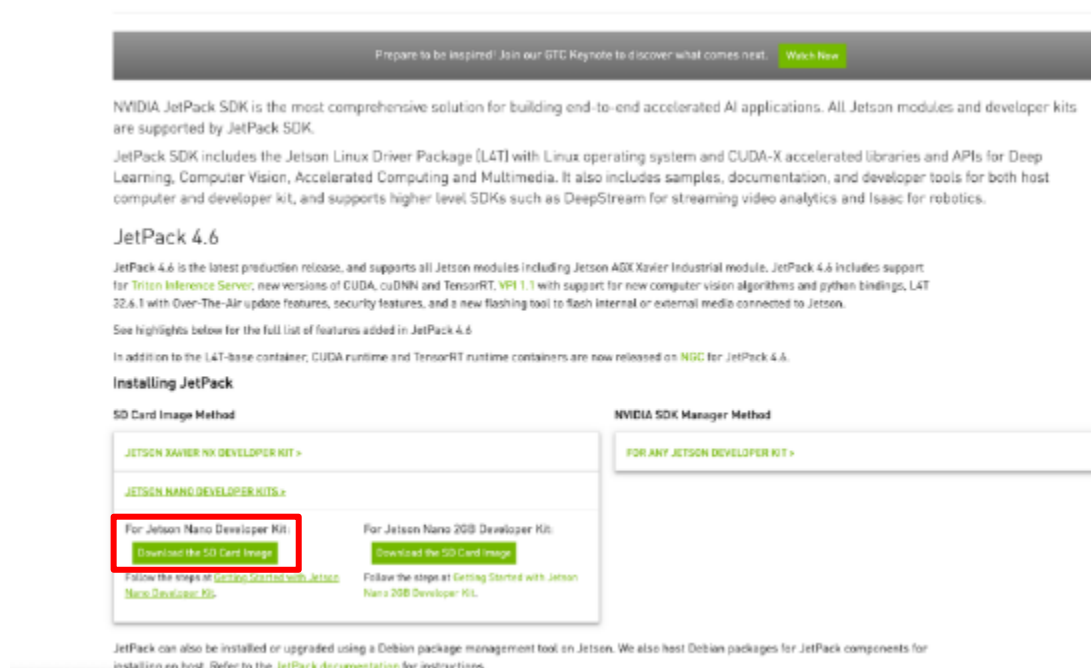


그림 43. JetPack 이미지 다운로드

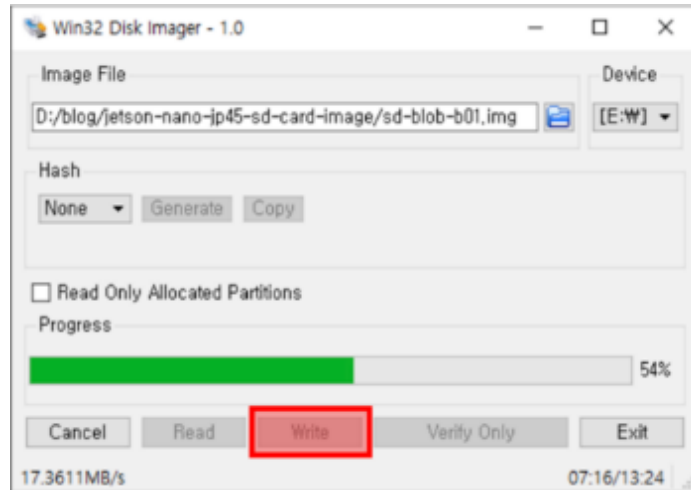


그림 44. SD 카드에 JetPack 이미지 쓰기

B. jetson-stats 설치

jetson-stats 는 Jetson Nano 의 종합적인 상태를 확인하는 도구이다. 설치 방법은 [표 47]에서 소개한다.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install python-pip
$ sudo -H pip install -U jetson-stats
$ sudo reboot
# jetson-stats 실행
$ jtop
```

표 47. Jetson-stats 패키지 설치

C. gdm3 삭제 및 lightdm 설치

Jetson nano 의 메모리 사용량을 줄이기 위해 삭제 및 설치를 한다. 설치 시 메모리 사용량이 1.4GB 에서 0.7GB 로 감소한다. 설치 방법은 [표 48]에서 소개한다.

```
$ sudo apt-get install lightdm
$ sudo apt-get purge gdm3
```

표 48. gdm3 삭제 및 lightdm 설치

D. 영상분석용 라이브러리 설치

a. OpenCV 4.5.4 with CUDA

Jetson Nano 에서 사용할 수 있는 OpenCV 다운로드 및 자동 설치 스크립트 파일을 다운로드한다. Jetson Nano 에서 OpenCV 를 설치하기 위해서는 [부록 1]의 E.b 항목에 설명한 방법으로만 설치할 수 있다. 그러나 컴파일 옵션 설정 방법이 복잡하기 때문에 Jetson Nano 의 환경에 맞게 설정한 스크립트 파일을 사용하면 더욱 편리하게 OpenCV 를 설치할 수 있다. 스크립트 사용시 시간은 오래 걸리지만 자동으로 필요한 라이브러리들을 설치하게 된다. 설치 방법은 [표 49]에 소개한다.

```
$ wget https://github.com/Qengineering/Install-OpenCV-Jetson-
Nano/raw/main/OpenCV-4-5-4.sh
$ sudo chmod 755 ./OpenCV-4-5-4.sh
$ ./OpenCV-4-5-4.sh
```

표 49. Jetson Nano 에 OpenCV 설치

b. PyTorch 1.8 + torchvision v0.9.0 다운로드

Jetson Nano 에서 PyTorch 와 torchvision 패키지를 설치할 때는 PyTorch 공식 홈페이지에 배포된 패키지를 사용하지 않고 NVIDIA 에서 제공하는 패키지를 설치해야 한다. 설치 시 필요한 종속 패키지도 함께 설치한다. 설치 방법은 [표 50]에서 설명한다.

```
$ wget
https://nvidia.box.com/shared/static/p57jwntv436lfrd78inwl7iml6p13fzh.wh
l -O torch-1.8.0-cp36-cp36m-linux_aarch64.whl
$ sudo apt-get install python3-pip libopenblas-base libopenmpi-dev
```

표 50. Jetson Nano 에 PyTorch, torchvision 패키지 다운로드

c. Cython, numpy, pytorch 패키지 설치

다운로드 받은 Jetson nano 용 PyTorch 패키지 및 Cython, numpy 패키지를 설치한다. 설치 방법은 [표 51]에서 소개한다.

```
$ pip3 install Cython
$ pip3 install numpy torch-1.8.0-cp36-cp36m-linux_aarch64.whl
```

표 51. Cython, numpy, torch 패키지 설치

d. torchvision 패키지 설치

torchvision 패키지를 설치하기 전에 종속 패키지를 먼저 설치한다. torchvision 패키지를 설치할 때는 pip 를 사용하는 것이 아니라 별도로 빌드하여 설치해야 한다. 설치 방법은 [표 52]에서 소개한다.

```
# torchvision dependencies 설치
$ sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev libavcodec-
dev libavformat-dev libswscale-dev

$ git clone --branch v0.9.0 https://github.com/pytorch/vision torchvision
$ cd torchvision
$ export BUILD_VERSION=0.9.0
$ python3 setup.py install -user
$ cd ../ # attempting to load torchvision from build directory will
result in import error
```

표 52. Jetson Nano 용 torchvision 패키지 설치

e. YOLOv5 실행용 필수 패키지 설치

[표 53]에 제시한 항목들을 YOLOv5 패키지 설치 목록에서 제거한 후 [표 54]과 같이 'pip install -r requirements.txt'로 필요한 라이브러리들 설치

```
# 다음 내용 requirements.txt 에서 제거
numpy>=1.18.5
opencv-python>=4.1.2
torch>=1.7.0
torchvision>=0.8.1
```

표 53. requirements.txt 내 설치 패키지 제거

```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install -r requirements.txt
```

표 54. YOLOv5 필수 패키지 설치

f. YOLOv5 소스코드 사용

YOLOv5 사용 테스트를 위해 [표 55]의 명령어를 실행한다.

```
# clone 한 repository 내부에 존재하는 image 에 대한 inference 수행하기
python3 detect.py --source ./data/images
# 연결된 webcam 을 통해 Inference 수행하기
python3 detect.py --source 0
```

표 55. YOLOv5 소스코드 실행 테스트

7. GitLab 프로젝트 리스트

현재 서비스 중인 영상분석 소스코드는 주차장 별로 GitLab 에 프로젝트를 생성하여 업로드하였다. 영상분석 서버에 실제 적용중인 브랜치는 굵은 글씨로 표시되어 있다. 다음과 같이 정리되어 있다.

주차장 (프로젝트) 이름	GitLab 프로젝트 그룹 / 이름	브랜치
안양 2 차 SKV1	WatchMile/Anyang_SKV1_inout	master yolov5_v5.0 yolov5_v6.0_v2
천호/강동역 공영주차장	WatchMile/Cheonho_Gangdong_inout	master Cheonho Gangdong
호반파크 2 관	WatchMile/hobanpark/hobanpark_inout	master local_test
안산 상하수도 공영주차장	WatchMile/ansan/ansan-parking-inout	master
보행자 인식	intern-2022-firsthalf/Pedestrian2Map	master
보행자 위치 표시	intern-2022-firsthalf/ClickPedestrian	master
차량 tracking	intern-2022-firsthalf/CarTracking	master
친환경 차량 판단 및 OCR (ALPR)	intern-2022-firsthalf/ALPR	master
장병희 인턴 주간보고서	intern-2022-firsthalf/ProjectReport	master
차량 tracking 을 통한 주차 판단	Yolov5_deepsort_trajectory	master