부산대학교
PUSAN NATIONAL UNIVERSITY

OpenCV/CNN 기반 페인팅로봇 작업경로 생성

담당 교수님 : 김원석

팀명 : 톰

201824451 김정호

201824581 정제영

201724601 최성렬

목차

1. 요구조건 및 제약 사항 분석에 대한 수정 사항

1.1 요구조건

1) Unity

개발된 알고리즘 및 모델을 활용하여 도장 로봇의 동작 과정을 시각화 하는데 Unity를 이용하여 시뮬레이터를 구성하는 것이 적절하다고 판단하여 기존에 test용도로만 사용할 예정이었던 Unity application을 시뮬레이터로 활용하게 되었다. 해당 과정에서 다음과 같은 요구 조건이 발생하였다.

기존 요구조건	◆ 현실의 도장 작업 환경과 유사한 환경 구현 - 도장면 및 도장로봇 Object - 도장면을 촬영할 카메라
추가 요구조건	◆ 시뮬레이터(Unity application)와 외부 모듈(Python)간 통신 및 스트리밍 - 해당 과정에서 서로 다른 두 프로세스 간 동기화가 필요

2) 영상 합성

넓은 범위의 객체 탐지를 위해, 기존에 수직으로 도장면을 촬영하는 방식에서 비스듬히 촬영 후 어라운드 뷰를 구현하는 방식으로 변경하였다. 또한 어라운드 뷰를 구현하기 위해 시뮬레이터 상의 도장 로봇 상하좌우에 장착된 4개의 카메라로 촬영한 영상을 이용하여 아래에서 위를 쳐다본 것 같은 영상을 구성하였으며 해당과정을 수행하기 위해 다음과 같이 요구조건 변경 및 추가가 발생하였다.

기존 요구조건	◆ 수직으로 촬영된 4대의 카메라를 이용하여 하나의 큰 이미지를 생성. ◆ 특징점 검출 알고리즘과 스티칭 함수를 이용한 변환행렬 계산 및 파노라마 이미지 생성
수정 요구조건	◆ 보다 넓은 범위를 촬영하기 위해 4대의 카메라를 50도 각도로 촬영 및 해당 이미지를 이용하여 어라운드 뷰 이미지 생성 - Checkerboard 이미지를 이용한 변환행렬 계산 - 원근 변환(Perspective Transformation)을 이용하여 원근 이미지를 수직한 평면 이미지로 변환 및 이미지 생성

3) Object detection

장애물 및 경계면, 기준선을 파악하기 위해 YOLO모델을 선 적용 후 Line detection을 수행하는 것으로 방법을 수정하였으며 edge detection의 정확도를 높이기 위해 Canny edge detection이나 Hough Transformation과 같은 전통적인 detection 방식에서 머신러닝 기반의 Line segment detection 모델을

도입하기로 결정했다. Texture는 어라운드 뷰로 재단된 이미지를 활용하는 것으로 결정했다.

기존 요구조건	◆ Edge detection후 YOLO 모델 사용
수정 요구조건	◆ YOLO모델로 먼저 객체 검출 후 Line segment detection으로 경계선 검출 ◆ 실제 도면 사진과 동영상을 활용 해 YOLO모델 학습 ◆ 머신러닝 기반의 Line segement detection 모델인 Airline모델 사용 ◆ Object detection은 시뮬레이터에서 실시간으로 동작

4) 작업 경로 생성 및 경로 이탈 감지 알고리즘 개발

기존에 작업 경로 생성은 용접선을 기준으로 생성하기로 하였으나, 용접선이 수평인 경우가 많아 지지대 및 가로 용접선을 활용하여 작업 경로를 생성하는 방식을 추가하였다. 경로 이탈 감지는 처음 생성된 작업 경로선과 실시간으로 탐지되는 기준선을 이용하여 이탈 유무를 결정한다.

기존 요구조건	◆ 용접선을 기준으로 작업경로 생성
수정 요구조건	◆ 지지대와 가로 용접선과 같은 선을 추가로 사용하여 작업 경로 생성 ◆ 작업 경로선과 검출된 용접선을 이용해 경로 이탈 유무 결정

1.2 기존 제약 사항 및 수정 사항

기존 제약사항 및 수정 제약사항

기존제약사항	수정 제약사항
경로이탈 감지에서 기준선인 용접선 소실 시 대책	실제 환경에서 용접선이 소실되는 경우는 거의 없기 때문에 제약사항에서 제외

기존 제약사항 별 대책 및 수정된 대책

기존제약사항	대책	수정대책
스티칭 과정에서의 왜곡 발생	이미지 스티칭 시 생성되는 시접선을 용접선, 장애물과 같은 객체를 회피하여 추정	상, 하 이미지와 좌, 우 이미지를 각각 합성 후 상/하 이미지를 기준으로 겹치지 않는 좌/우 이미지를 합성
온 습도 및 조명환경의 변화에 의한 합성 이미지 불균형	명도 보정을 통한 이미지 선명도 조절	경로생성 알고리즘의 검증을 위해 Unity 시뮬레이션을 활용하므로 광원을 일정하게 설정해도 무방
모델 및 경로생성 알고리즘의 검증 어려움	Unity를 이용한 시뮬레이터 만들기	변경없음

2. 설계 상세화 및 변경 내역

2.1 시뮬레이터

시뮬레이터는 실제 도장 환경과 유사한 환경을 갖추어야 하며, 적당한 차량 asset을 다운로드 하여 도장로봇으로 하고 Cube에 texture를 입혀 영상 합성에 필요한 Checker board 패턴의 판 및 도장면과 유사한 간이 도장판을 구성하였다.

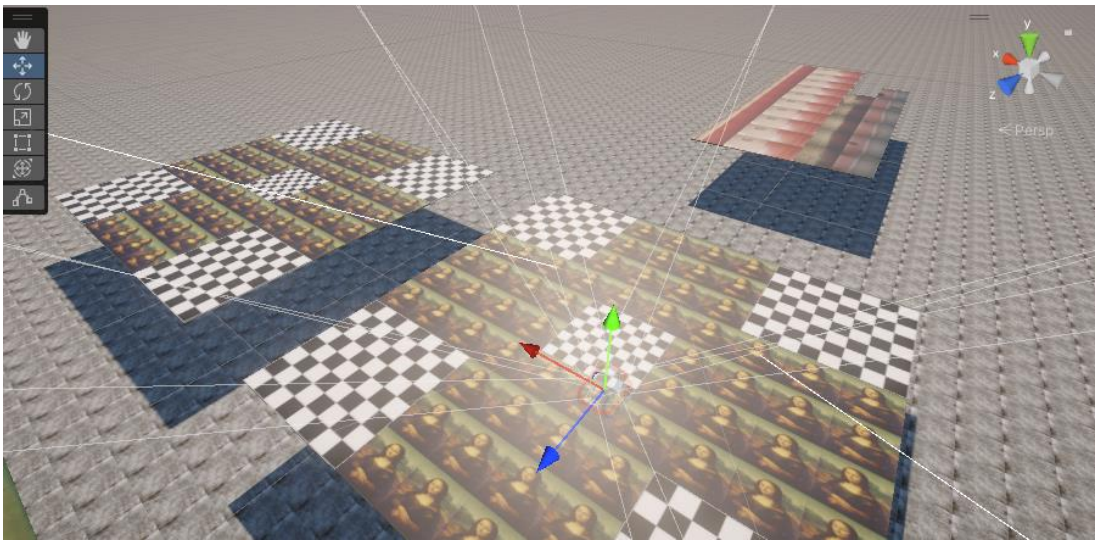

Figure 1 Unity Editor 화면

시뮬레이터는 실행 시 영상 합성을 위한 외부 모듈(Python)을 실행하며, 시뮬레이터의 도장 로봇 상하좌우의 카메라에서 화면을 렌더링 하여 이미지로 저장한다. 또한 외부 모듈과 소켓 통신을 통해 외부 모듈과 동기화 및 렌더링 된 이미지 정보를 외부 모듈에 전송하고 Python에서 합성된 이미지를 시뮬레이터 화면에 스트리밍 한다.

2.2 영상 합성

영상 합성의 경우 다음의 3단계를 거쳐 이루어진다.

1) 이미지 불러오기

시뮬레이터에서 렌더링된 4장의 이미지를 Python으로 읽어 온다.

```
host, port = "127.0.0.1", 25001
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((host,port))

sock.setblocking(True)

sock.sendall(teststring.encode("UTF-8"))


while True:
    receivedData = sock.recv(1024).decode("UTF-8")
    if receivedData != "" and temp!= receivedData:

        #이미지 읽어오기
        receiveImgN = cv2.imread(receivedData + "N.png")
        receiveImgS = cv2.imread(receivedData + "S.png")
        receiveImgE = cv2.imread(receivedData + "E.png")
        receiveImgW = cv2.imread(receivedData + "W.png")
```

Code 1 Socket을 이용하여 이미지 불러오기 코드

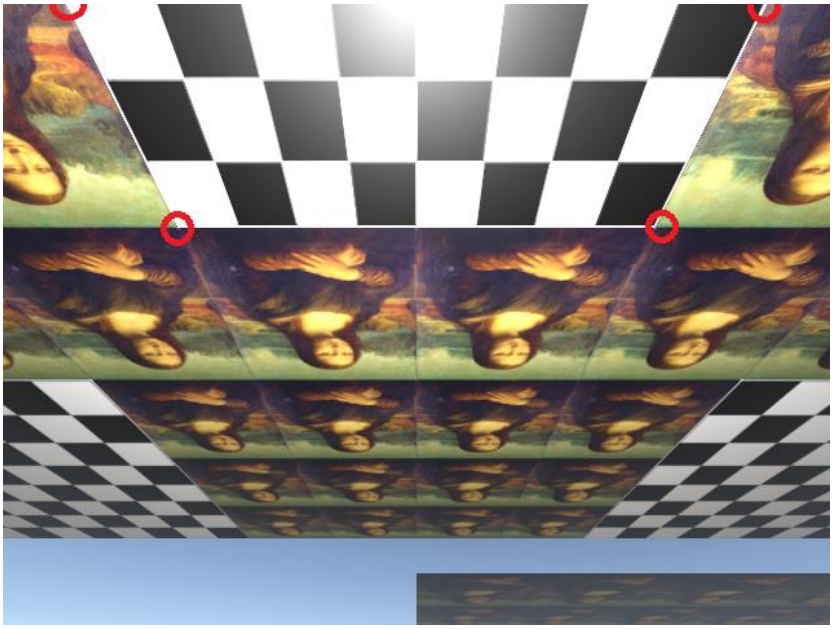

Figure 2 원본 이미지

2) 이미지 원근 변환

원근 변환의 변환 행렬을 계산하기 위해 기준이 될 4개의 좌표(원본 이미지의 표시된 지점)를 설정한다.

Checkerboard 한 칸의 크기를 15픽셀로 설정하고 대입할 이미지 좌표를 정한다.

원본 이미지 좌표와 대입할 이미지 좌표를 이용하여 3by3 변환행렬을 계산한다.

```python
def warpping(img): #원근변환 함수


    # 이미지의 높이, 너비 픽셀 좌표
    h = img.shape[0]
    w = img.shape[1]



    #대입할 이미지 중심 설정
    shift_w = w/2
    shift_h = 0

    # 원본 이미지 좌표 [x,y] (좌상 좌하 우상 우하)
    #화각 87도 기준
    ori_coordinate = np.float32([[50, 0], [135, 172], [590, 0], [505, 172]])

    #대입할 이미지 좌표 (좌상 좌하 우상 우하) 한칸이 15
    warped_coordinate = np.float32([[shift_w-60, shift_h], [shift_w-60, shift_h+40], [shift_w+60, shift_h], [shift_w+60, shift_h+40]])

    #3X3 변환 행렬 생성
    Matrix = cv2.getPerspectiveTransform(ori_coordinate, warped_coordinate)

    #원근 변환
    warped_img = cv2.warpPerspective(img, Matrix, (w, h))
    return warped_img
```

Code 2 원근 변환 코드

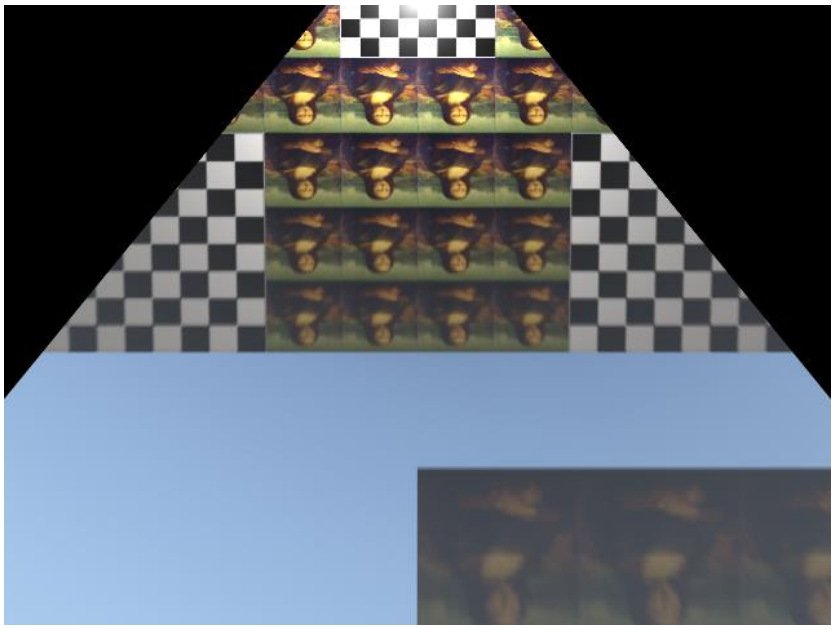

Figure 3 원근 변환 이미지

3) 변환된 이미지 합성

상하좌우 이미지 중 상, 하 및 좌, 우 이미지를 합성 한 뒤 상/하 이미지를 기준으로 겹치지 않는 부분끼리 합성한다.

```
# 겹치는 부분
white_color = (255,255,255)
mask = np.zeros((600,600,3),dtype = np.uint8)
pt1 = np.array([[0,0],[300,240],[600,0],[360,300],[600,600],[300,360],[0,600],[240,300]], np.int32)
mask = cv2.polylines(mask,[pt1], True, white_color)

white_star = cv2.fillPoly(mask, [pt1], white_color)
black_star = cv2.bitwise_not(white_star)

dup_WE = cv2.bitwise_and(background_WE, white_star)
dup_NS = cv2.bitwise_and(background_NS, white_star)

# 겹치는 부분 제외
x_star = cv2.bitwise_and(b, black_star)

# 기존 결과
#cv2.imshow('b', b)

# 가중치 합성결과
dst_NS = cv2.bitwise_or(dup_NS, x_star)
dst_WE = cv2.bitwise_or(dup_WE, x_star)

# Fill 0 values in dst_NS with corresponding values from dst_WE
dst_NS[dst_NS == 0] = dst_WE[dst_NS == 0]
```

Code 3 이미지 합성 코드

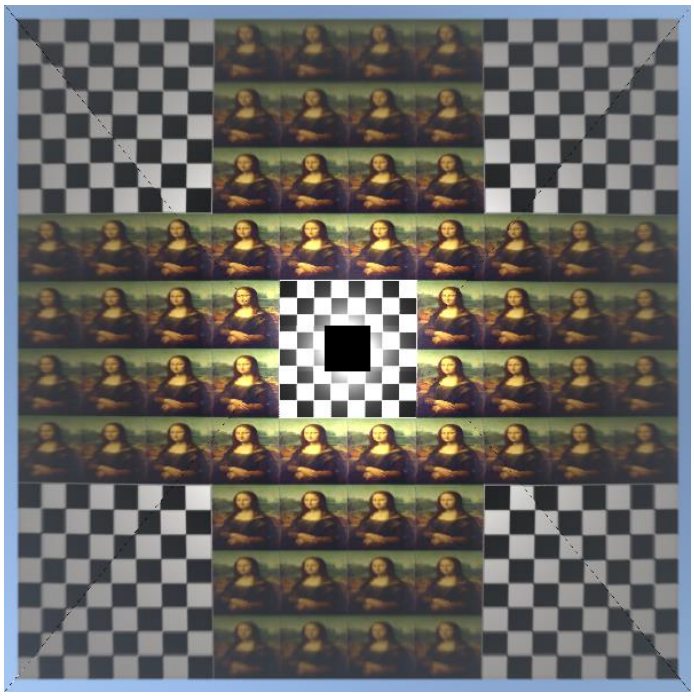

Figure 4 결과 이미지

2.3 detection

Detecting을 위해 필요한 과정은 다음과 같다.

1) 이미지 전처리

학습 데이터로 활용할 실제 도면 사진 및 영상이 부족하다. detection에 필요한 모델 성능 확보를 위해서 학습 데이터의 양을 늘릴 필요가 있다. 따라서 기존에 확보된 이미지들을 회전, 반전, 자르기 등을 이용해 데이터를 늘리는 방법을 택했다. 또한 학습을 위해 용접선은 수평 또는 수직이 되게끔 이미지를 회전한다. 이러한 방법으로 Train set 및 Test set을 확보한다.

2) 이미지 라벨링

YOLO모델에 전이 학습을 하기 위해 이미지 라벨링이 필요하다. 데이터셋을 만들기 위해 이미지에서 Bounding Box를 지정하여 라벨링을 수행한다. Bounding Box의 경우 정규화된 수치좌표로 나타내어야 한다. 이러한 과정을 수행하기 위해 이미지 라벨링 툴 labelimg를 이용해서 이미지 라벨링을 수행한다. 총 6가지 클래스로 나누어 라벨링 했으며 종류는 다음과 같다.

1. welding line - 용접선
2. paint border - 페인트 경계선
3. carrier border - 지지대 경계선
4. obstacle - 장애물
5. border – 도장면 경계선
6. hole – 구멍

3) YOLOv5s6 학습

Object detection을 위해 YOLO모델을 사용한다. 경량화 모델인 YOLOv5s6을 사용하여 실시간성을 확보하고자 한다.

4) Airline: Line segment detection

YOLO 모델을 이용해 나온 결과를 기반으로 Line segment detection을 수행한다. 먼저 YOLO로 경로 생성 기준선 영역을 검출한다. 이후 검출된 영역 안에서 머신러닝 기반의 Line segment detection 모델을 사용해 직선을 검출한다. Line segment detection 모델은 직선 정확도가 높은 AirLine 모델을 사용했다. 아래는 AirLine 모델의 detection 과정을 그림으로 나타낸 것이다.

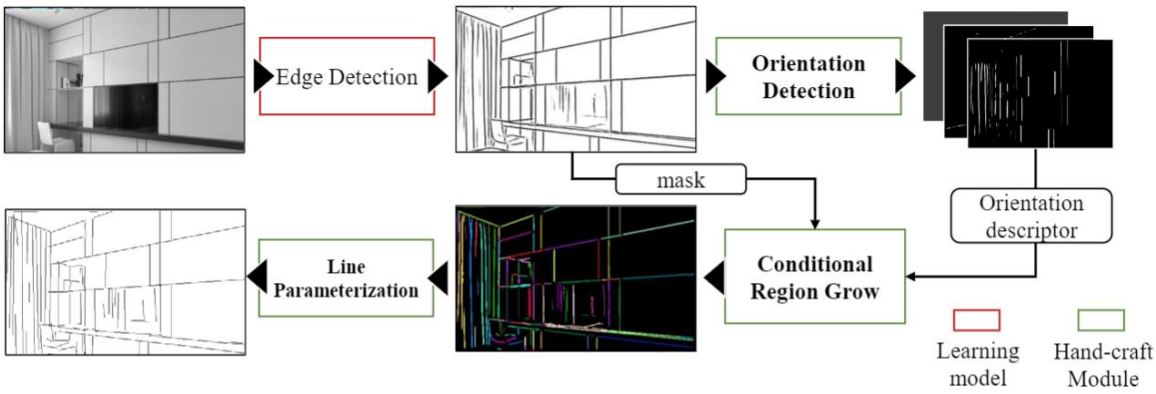

Figure 5 AirLine 과정

3. 갱신된 과제 추진 계획

<table>
<tr><td colspan="2">5월</td><td colspan="5">6월</td><td colspan="4">7월</td><td colspan="5">8월</td><td colspan="4">9월</td></tr>
<tr><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td></tr>
<tr><td colspan="3">관련 기술 지식 학습</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td colspan="6">데이터 수집 및 전처리</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td colspan="3">Calibration 구현</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td colspan="5">Edge detection 모델 구현 및 학습</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">경로 생성 알고리즘 개발</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">중간보고서 작성</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="6">시뮬레이터 구현</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">안정성 및 성능 평가</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">문제점 파악 및 오류 수정</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">최종보고서 작성 및 발표 심사 준비</td></tr>
</table>

4. 구성원 별 진척도

조원	역할
김정호	Unity 시뮬레이터 구현
정제영	YOLOv5 로 도장면 분류 모델 개발 중
최성렬	OpenCV 로 도장면 영상 합성 구현
공통	데이터 라 벨 링 , 보고서 작성

5. 보고 시점까지 과제 수행 내용 및 중간 결과

5.1 시뮬레이터

도장 환경과 유사한 환경을 조성하기 위해, 상부에 넓은 판을 띄워 놓는 식으로 object를 구성하였으며, "`"버튼을 클릭할 시 콘솔창을 불러와 소켓통신이 제대로 이루어지고 있는지 확인할 수 있도록 구성하였다. 화면 하부 3개의 버튼의 기능은 다음과 같다.

1. First person : 메인 카메라의 시점을 1인칭으로 한다.

2. Capture : 차량에 설치된 4개의 카메라가 보고 있는 장면을 렌더링 하여 저장한다.

3. Third person : 메인 카메라의 시점을 3인칭으로 한다.

Capture 버튼 위의 Toggle 버튼은 활성화 시 스트리밍을 수행하여, 합성 결과를 화면 우상단에 출력한다.

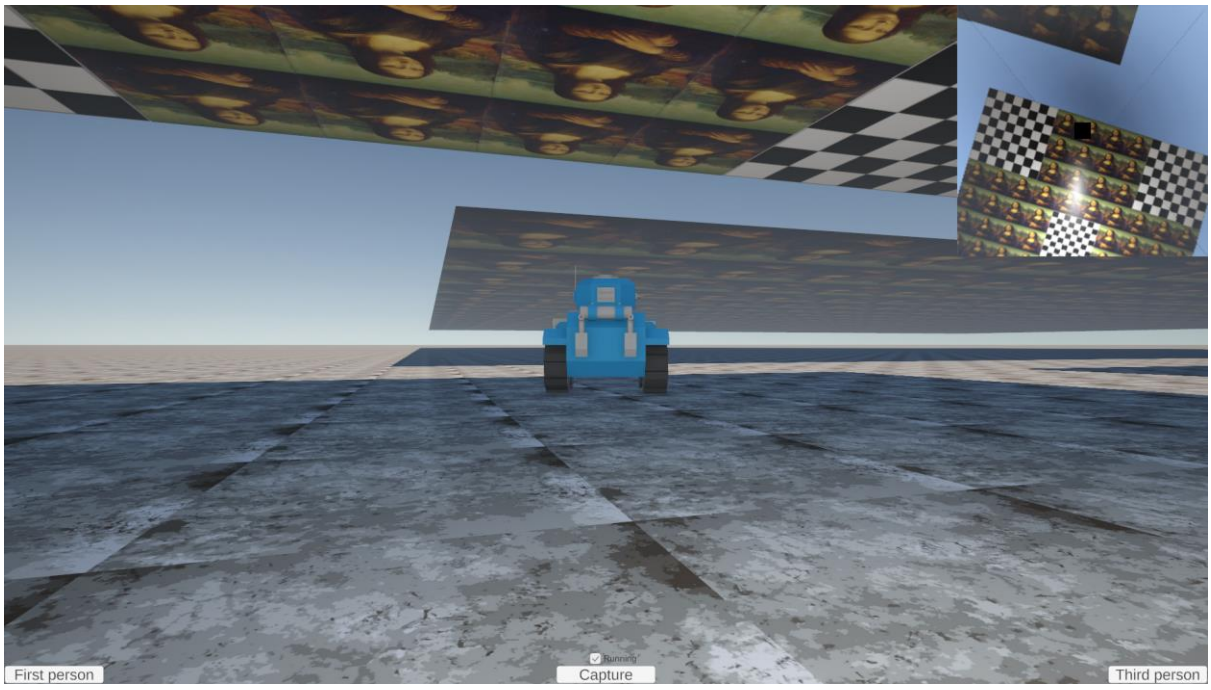

Figure 6 시뮬레이터 실행 화면

5.2 영상 합성

2.2에서 설명한 과정을 거쳐 around view를 구성하게 된다.

먼저 최초 1회 원근행렬을 구성하여, 구성된 행렬을 이용해 이미지를 Orthogonal하게 편다. 펼쳐진 이미지를 겹치지 않게, 맹점을 계산하고, 이격을 설정하여 합성한다.

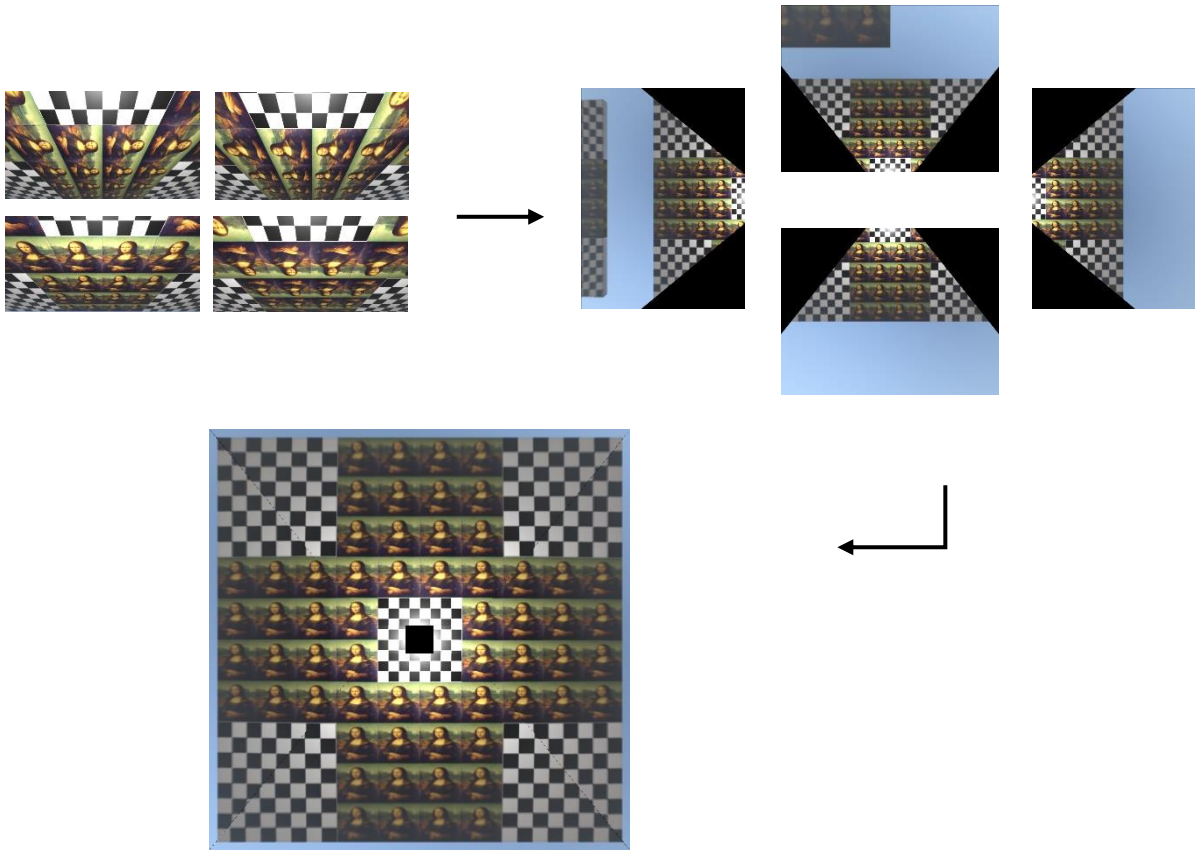

Figure 7 이미지 합성 과정

5.3 detection

YOLOv5s6모델 및 학과에서 제공되는 AI서버를 이용하여 모델 학습을 진행한다. AI서버를 이용한 학습을 위해서 Docker 이미지가 필요하여 Docker 파일을 작성하였다.

학습을 위해 제공된 이미지와 영상 데이터를 이용해 약 700장 정도의 라벨링을 진행하였으며 그 중 약 500장을 이용해 학습을 진행하였다.

학습 결과는 다음과 같다.

아직 데이터 셋이 부족하여 따로 Validation set을 두지 않고 test data set 안에서 Validation set을 두었다.

Figure 8 YOLO detecting 이미지1

앞의 두 이미지는 traing set에 포함되지 않았던 이미지이며 마지막 이미지는 지지대의 detecting을 보여주기에 적절한 이미지가 부족하여 traing set에 포함된 이미지를 사용하였다.

위의 사진들과 같이 전반적으로 준수한 성능을 보이나

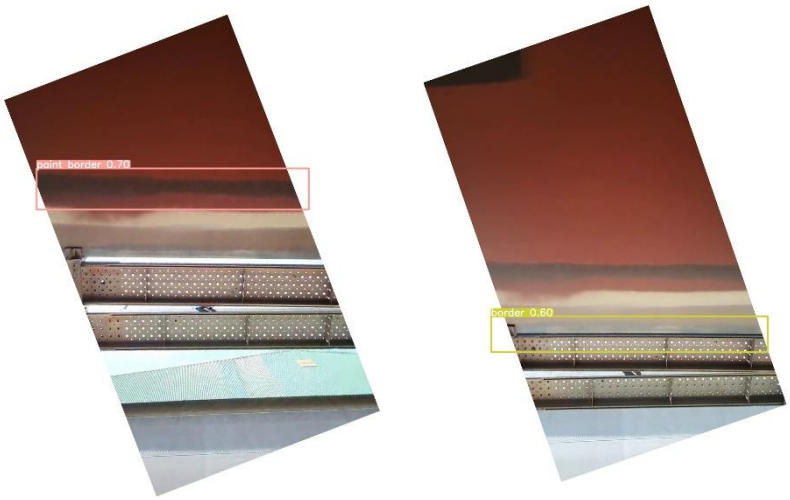

Figure 9 YOLO detecting 이미지2

test set의 이미지 중 일부는 위의 그림과 같이 detecting 시 불안정한 성능을 보이기도 한다.

그러나 데이터 셋을 증가시킬수록 YOLO에서 제공하는 지표가 더 안정화되는 것을 확인할 수 있으며 에폭수 및 배치 사이즈의 조정 및 전통적인 머신러닝의 기법들을 아직 적용해 보지 못했기 때문에, 개선의 여지가 상당 부분 남아 있다.

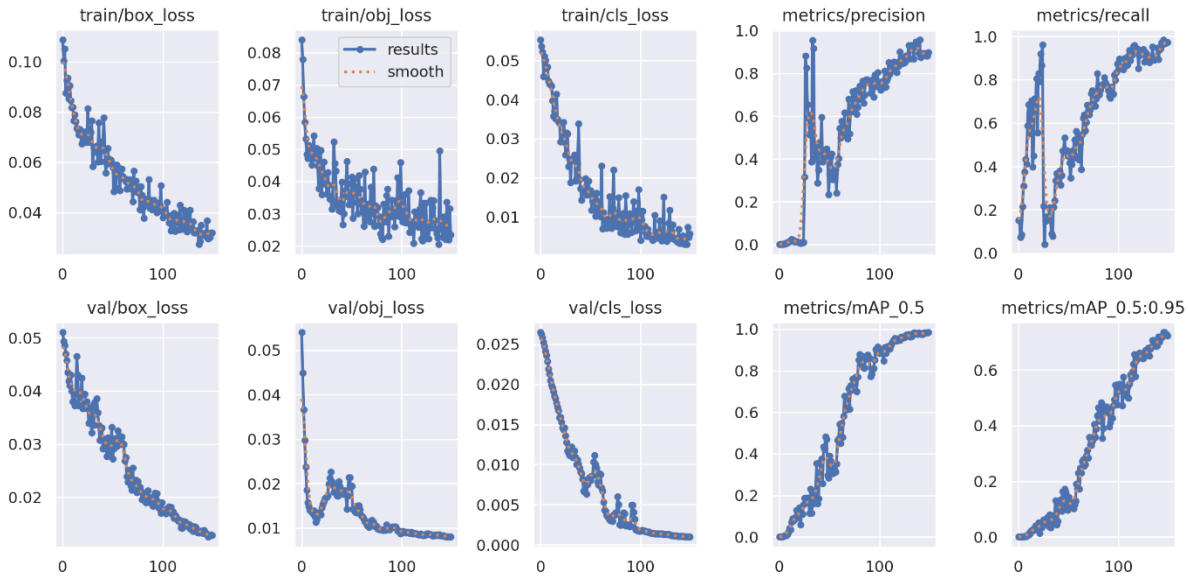

Figure 10 이미지 약 100장을 이용해 구성한 YOLO 모델의 성능 평가 지표

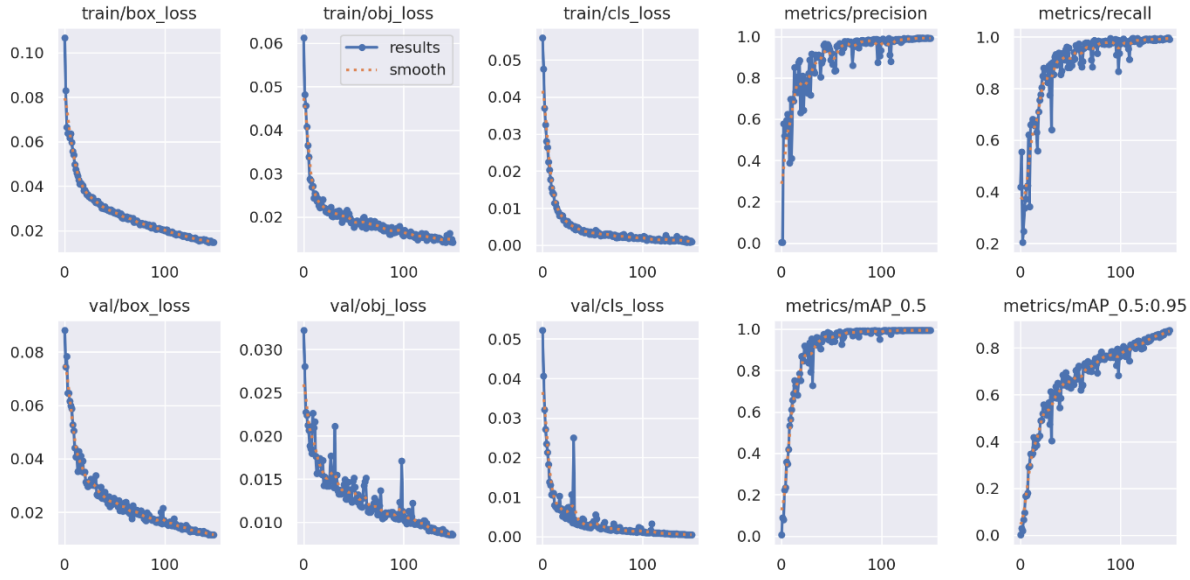

Figure 11 이미지 약 500장을 이용해 구성한 YOLO 모델의 성능 평가 지표