

# OpenCV 기반 자율주행 페인팅 로봇



김정호

정제영

최성렬

지도교수 김원석

---

## 목 차

1. 서론.....	1
1.1. 연구 배경.....	1
1.2. 기존 문제점 .....	1
1.3. 연구 목표.....	2
1.3.1. aroundview .....	3
1.3.2. YOLO.....	3
1.3.3. line detecting.....	3
1.3.4. Unity .....	3
2. 배경 지식.....	4
2.1. OpenCV .....	4
2.2. YOLOv8 .....	4
2.3. Airline.....	6
2.4. curve fitting .....	7
3. 연구 내용.....	8
3.1. aroundview.....	8
3.2. YOLO .....	11
3.2.1. 학습 데이터 수집 및 데이터셋 구성.....	11
3.2.2. 학습 데이터 라벨링 .....	11
3.2.3. YOLO 학습 결과 .....	12
3.2.4. YOLO 학습데이터 증강 .....	12
3.3. line detecting .....	14

---

3.3.1. Unity .....	19
4. 연구 결과 분석 및 평가 .....	20
4.1. 설계 변경 내역 .....	20
4.2. 연구 결과 .....	20
5. 결론 및 향후 연구 방향 .....	21
5.1. 결론 .....	21
5.2. 향후연구 .....	22
5.2.1. 실제 도장환경 테스트 베드 구축 .....	22
5.2.2. 작업경로 생성 모듈 및 작업차량 위치 추적 프레임워크 개발 .....	22
6. 구성원별 역할 및 개발 일정 .....	22
6.1. 개발일정 .....	22
6.2. 구성원 별 역할 .....	23
7. 참고 문헌 .....	23

## 1. 서론

### 1.1. 연구 배경

최근 조선업계에서는 업황의 개선에도 기술자의 리턴이 거의 없어 외국인 노동자로 채울 수밖에 없는 큰 어려움을 겪고 있다. 이러한 인력난을 타개하기 위해 조선3사는 자동화를 추진하고 있다.

## 조선 3사, 심각한 인력난 '자동화'로 극복한다

기사입력 : 2023년03월10일 17:21 | 최종수정 : 2023년03월10일 17:21



가 + 가 - 프린트

현대중공업, 2030년까지 스마트조선소 목표  
대우조선, 협동로봇 확대로 생산성 향상  
삼성중공업, 공정별 자동화...스마트야드 연구도

Figure 1. 조선3사 업황 뉴스

스마트 야드 라는 개념의 등장으로 선박의 설계, 생산, 조달, 관리 등 다양한 분야에 자동화를 비롯한 IT기술이 접목되어 조선소를 스마트 야드화 하고 있다. 조선소 내부에서 많은 연구 및 자동화가 이루어지고 있지만, 도장 작업은 자동화 비율이 높지 않다. 이 같은 상황에 작업 차량의 무인화를 이룬다면 작업 효율성 증대, 작업 안정성 강화 및 자동화를 통한 산업재해 감소 또한 기대해볼 수 있다.

따라서 본 과제에서는 도장 로봇의 작업 자동화를 위해 비전 인식과 보정 기술을 통해 블록 하부 도장 작업 자동화를 진행해보고자 한다.

### 1.2. 기존 문제점

도장 작업은 선체 곳곳에 페인트를 분사하여 칠하여 부식 따위를 막는 작업을 통칭한다. 그 중 블록 하부 도장 작업은 숙련된 전문인력이 직접 작업 차량을 조작하는 형태로 상당한 시간이 소모되며, 작업 시 발생하는 분진으로 인해 작업자의 건강, 안전 문제가 존재한다.

또한 다년간의 숙련이 필요하나, 장기간 기술을 익혀야 할 한국 조선 산업의 미래를 이끌어갈 MZ세대(1980~2000년대 초반 출생자) 생산 인력이 조선소에서 대거 이탈하고 있어 조선소 내의 고령화가 빨라지면서 우리 조선업의 중장기 미래도 불투명해지고있다. 특히 도장

작업의 경우 인력 총원이 가장 미비하여 2030세대의 비율이 약 20%에 불과하다.

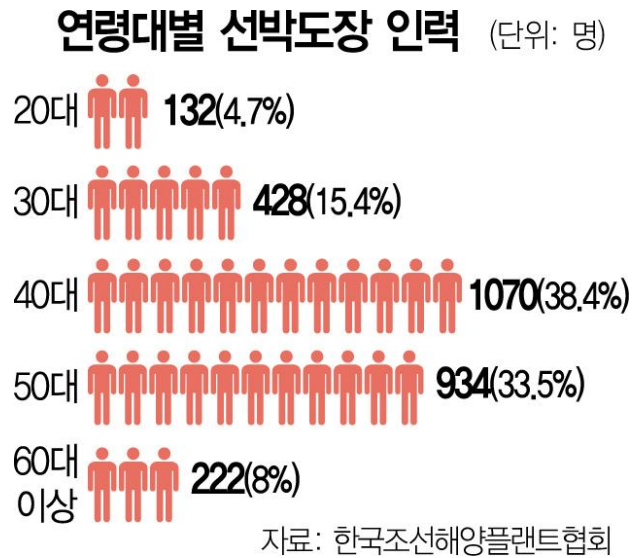


Figure 2. 연령대별 선박도장 인력

### 1.3. 연구 목표

본 졸업 과제는 RGB카메라 영상을 딥러닝 기반 모델 및 OpenCV를 활용하여 하부 도장로봇의 자율주행 실제 도장 작업에 적용 가능한 제어 신호를 생성하고 Unity 시뮬레이터를 구현해 적용하여 실제 도장작업에 사용 가능한 해결방안을 제시하는 것을 목표로 한다.

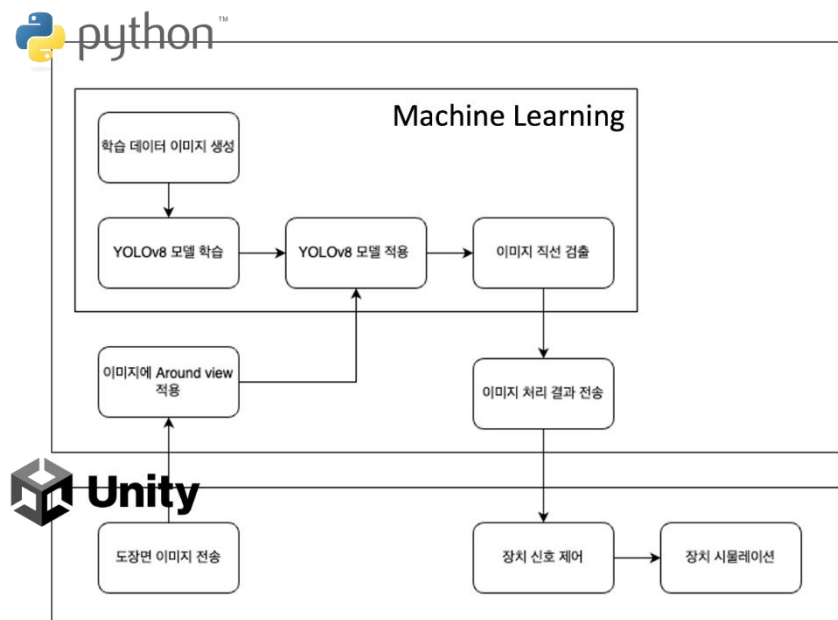


Figure 3. 시스템 구성도

---

### 1.3.1. aroundview

도장면에는 다양한 object가 존재하며, 카메라 하나만을 사용하여 도장면의 정보를 수집하기에는 도장면과 로봇의 간격이 너무 가깝다. 따라서 정보의 광범위한 수집을 위해 데이터의 수집을 위해 4개의 카메라를 활용해 상하좌우 이미지를 촬영 후 aroundview 영상을 합성한다.

### 1.3.2. YOLO

YOLO는 다중클래스 탐지를 지원하여 도장면의 다양한 정보를 효과적으로 인식할 수 있다. 또한 이미지를 한 번만 처리해서 실시간으로 객체를 감지할 수 있는 빠른 속도로 동작하여, 객체의 위치와 클래스를 신속하게 예측할 수 있다. 이러한 특징을 토대로 도장면의 다양한 객체를 파악하되 실시간성이 요구되는 본 과제를 수행하는데 적합하다.

따라서 우리는 삼성중공업 스마트 야드의 실제 도장면 영상을 기반으로 YOLO모델을 학습한다.

### 1.3.3. line detecting

Canny, Hough Transform등의 고전적인 line detecting 방법 대신 dexiNed기반의 Airline을 활용하여 line을 detecting한다. 이후 학습된 YOLO모델의 detecting 결과를 이용하여 클래스를 구분하고 그 중 적절한 선을 이용하여 기준선을 검출한다.

### 1.3.4. Unity

앞선 과정을 실제 환경 또는 테스트베드 구축을 통해 제시하기에는 비용 및 시간적인 한계로 인해 사실적인 물리엔진을 제공하는 Unity를 활용하여 실제 환경과 유사한 시뮬레이터를 구현한다.

시뮬레이터에서는 검출된 정보를 이용해 본 과제의 실제 하부 도장 환경 적용 가능성을 시사한다.

---

## 2. 배경 지식

### 2.1. OpenCV

OpenCV는 C/C++로 개발된 실시간 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리이며 실시간 이미지 프로세싱에 중점을 둔 라이브러리이다. 또한 기존의 영상 처리 라이브러리들과 다르게 오픈 소스로, 영상 관련 라이브러리로서 표준에 가까운 지위를 가지고 있다. 다양한 영상처리 API를 지원하며 본 과제에서는 원근변환 행렬 구성 및, 이미지 합성을 위해 OpenCV를 사용한다.

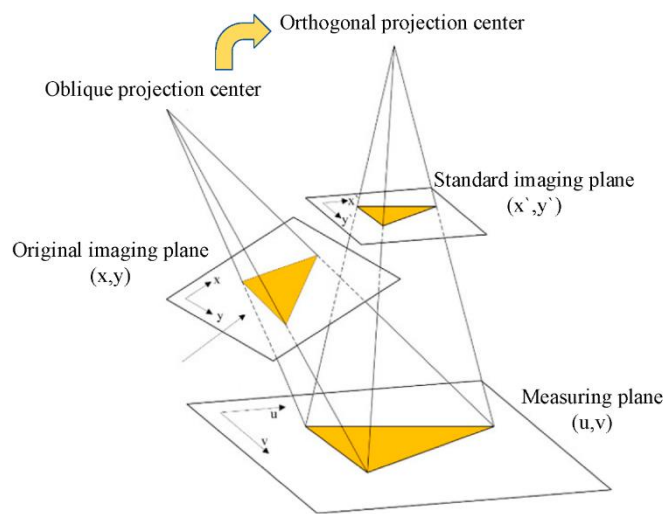


Figure 4. 원근 변환

### 2.2. YOLOv8

본 과제에서 실시간 객체 탐지를 위해 YOLO모델을 요구한다. YOLO모델은 다양한 버전이 있는데 범용적인 적용을 위한 버전으로 YOLOv5와 YOLOv8이 있다. YOLOv5는 Pytorch에 내장되어 보다 사용이 쉽고 2020년 6월에 출시되어 안정성이 검증되어 있다는 장점이 있으나, 비교적 최근에 출시된 YOLOv8이 속도 및 정확성 면에서 더 좋은 모습을 보여 YOLOv8모델 사용을 결정했다.

## Object Detection Performance Comparison (YOLOv8 vs YOLOv5)

Model Size	YOLOv5	YOLOv8	Difference
Nano	28	37.3	+33.21%
Small	37.4	44.9	+20.05%
Medium	45.4	50.2	+10.57%
Large	49	52.9	+7.96%
Xtra Large	50.7	53.9	+6.31%

\*Image Size = 640

Figure 5. YOLOv5/v8간 성능 차이

YOLOv8은 출시된 YOLO버전 중 가장 최근에 나왔으며 2023년 1월 Ultralytics에서 개발되었다. YOLOv5와 비교했을 때 변경점은 다음과 같다.

- C3 모듈을 C2f 모듈로 교체
- Backbone에서 첫번째 6x6Conv를 3x3Conv으로 교체
- 두 개의 컨볼루션 레이어 삭제
- ConvBottleneck에서 첫번째 1x1 Conv를 3x3 Conv로 교체
- 분리된 head 사용 및 objectness 분기 삭제



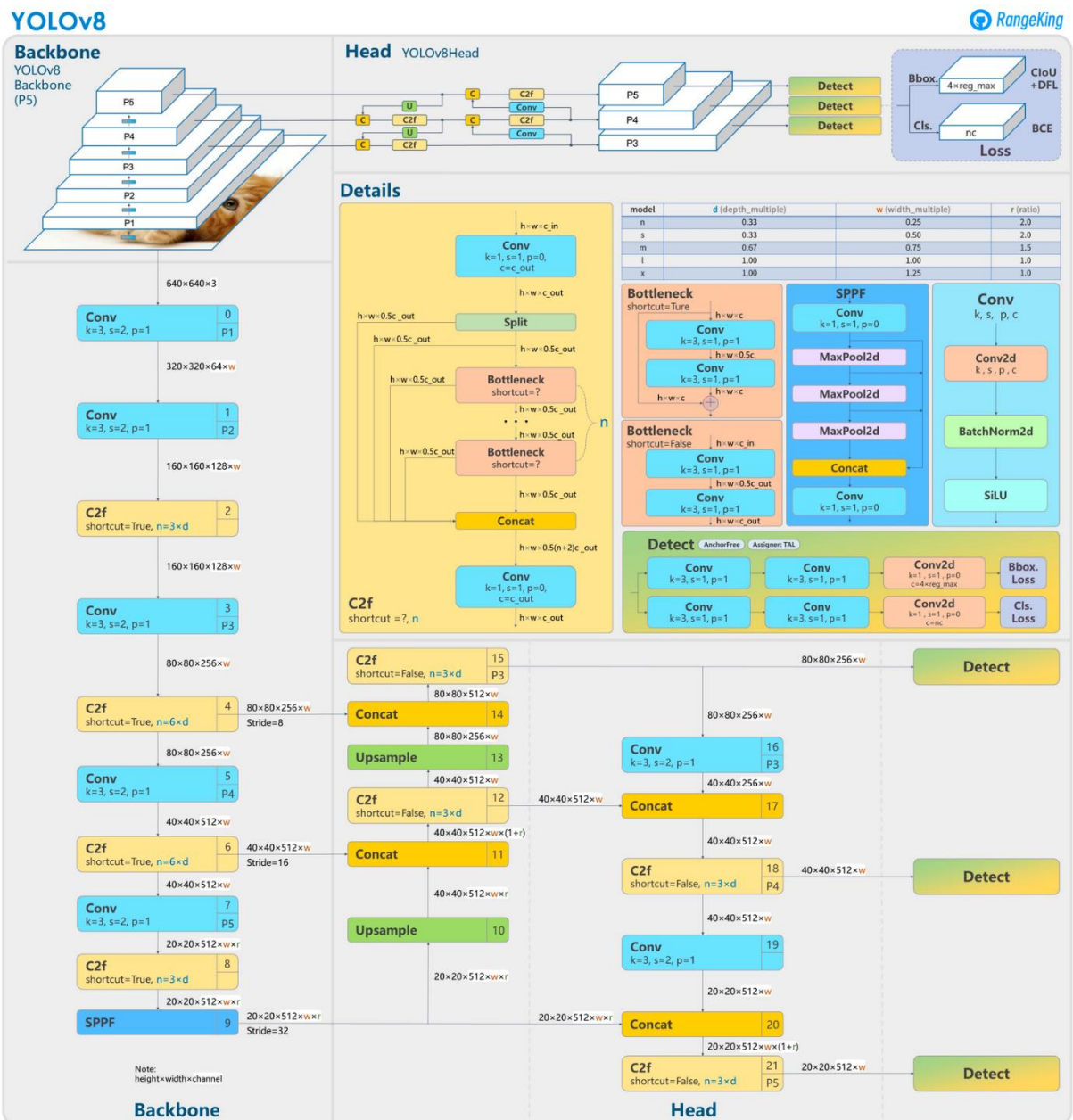


Figure 6. YOLOv8 모델 구조

YOLOv8은 nano부터 Xlarge 모델까지 제공되며, 본 과제에서는 성능 및 실시간성이 균형을 이루는 YOLOv8s 모델을 채택하였다.

### 2.3. Airline

Line detecting에는 Canny, Sobel 등과 같은 전통적인 방식과 LSD, LCNN 등 딥러닝을 기반으로 하는 방식 등 많은 방법이 존재하지만 이번 과제에서는 Airline line detection을 사용해 보았다.

Airline은 Unet과 DexiNed 모델을 기반으로 하는 line detection 방식으로 Canny, Sobel, LSD, LCNN 같은 기존 방식들 보다 뛰어난 성능을 보인다.

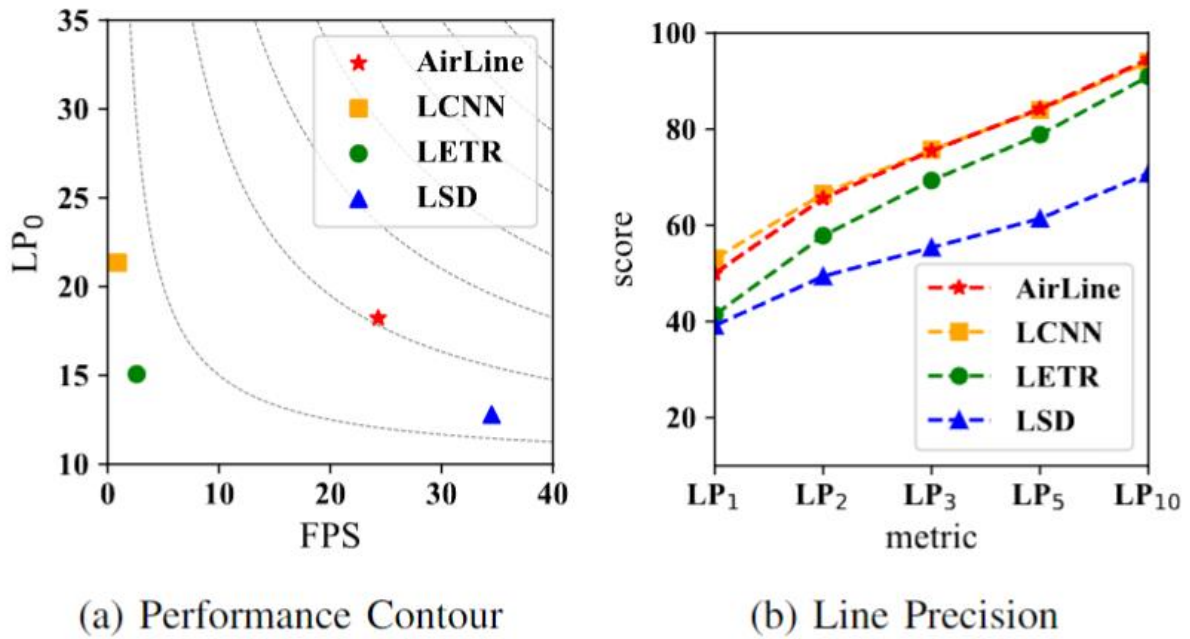


Figure 7. Airline의 상대적 성능

Dexined 모델을 통한 edge detection 후 방향감지, 조건부 영역 증가 및 선 매개변수화를 통해 학습가능한 edge 기반 line detection 구조로서 YOLO를 통해 검출된 bounding box를 이용, 기준선 및 진입점을 특정하기위해 사용했다.

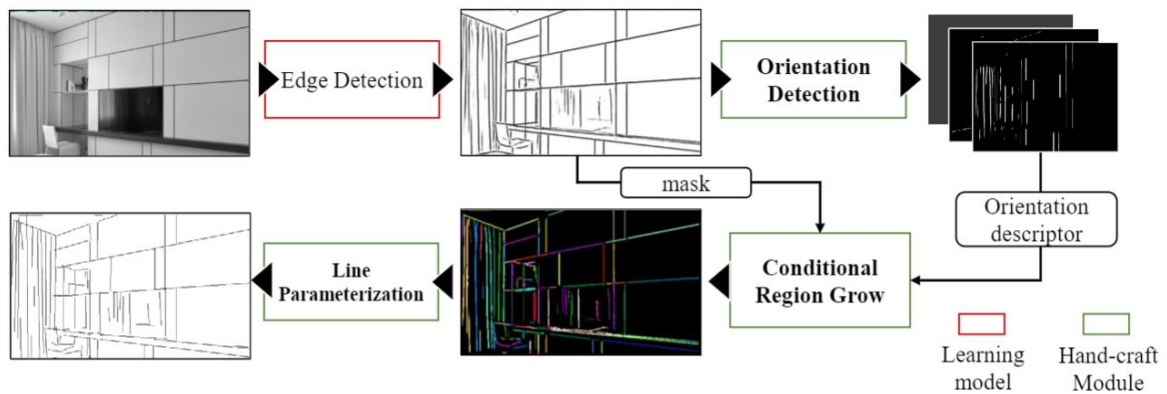


Figure 8. Airline line detection 과정

## 2.4. curve fitting

Curve\_fitting은 이산적으로 주어진 데이터를 가장 적절히 표현할 수 있는 함수식을 추정하

---

는 공정이다. 대표적으로 최소제곱 회귀분석법과 보간법으로 나뉘는데, 데이터의 경향을 파악하려는 경우 회귀분석법을 사용하고, 주어진 모든 점을 포괄하는 함수를 구하고자 할 경우 보간법을 사용한다. 최소제곱 회귀분석법을 이용할 경우 선형의 함수를 구할 수 있으며, 본 과제에서 선을 검출하고자 하므로 해당 방법을 사용하였다.

최소제곱 회귀분석법은 오차를 최소화하는 미지수를 찾는 방식이며 종속변수  $Y$ 와 하나이상의 독립변수  $X$ 간의 관계를 모델링 하는 것이다. 이 관계를 선형 함수로 나타내면 다음과 같다.

$$Y_i = b_0 + b_1 X_i + \epsilon_i$$

여기서 각각

- $Y_i$ : 종속변수
- $X_i$ : 독립변수
- $b_0$ : 모델의 절편
- $b_1$ : 독립변수  $X$ 의 계수
- $\epsilon_i$ : 오차 항

이며 최소 제곱법을 통해 모델 파라미터  $b_0, b_1$ 을 추정하기 위해서 오차항이 최소가 되어야 하고, 따라서 식을 다시 쓰면 다음과 같다.

$$\text{Min} : S^2 = \text{Min} : \sum_{i=1}^n \epsilon_i^2 = \text{Min} : \sum_{i=1}^n (y_i - b_0 - b_1 x_i)^2$$

그리고 이 식을 최소화시키는  $b_0, b_1$ 를 찾기 위해 편미분을 이용하여 도함수 편미분 행렬을 이용하여 이계도함수를 구해 아래로 볼록한 극점을 찾아, 오차항이 최소가 되는 함수를 찾을 수 있다.

### 3. 연구 내용

#### 3.1. aroundview

1대의 카메라로 도장면의 영상 정보를 수집하기에는 도장면과 도장로봇간의 거리가 가까워

---

충분히 많은 범위를 촬영할 수 없다. 따라서 4대의 카메라를 도장로봇 전후좌우에 부착하여 각각 영상을 촬영하고, 4개의 영상을 합성하여 사용했다. 4대의 카메라를 활용해 360도 모든 영상정보를 수집하기 위해 화각 약 160도 정도의 어안렌즈를 도장로봇과 50도 각도를 이루게 설치하여, 사각지대를 최소화하였다. 수집한 영상을 용이하게 처리하기 위해 4개의 영상을 aroundview로 합성하여 orthogonal한 영상을 구성하였다.

OpenCV에서 이미지 stitching 함수를 제공하여, 이를 이용하면 쉬이 aroundview 이미지를 구성할 수 있을 걸로 생각하였으나, 도장면의 특성상 이미지 합성의 기준점이 될 특징점이 전혀 특정되지 않아 stitching 함수를 사용할 수 없었다. 따라서 카메라가 고정되어 있다는 사실을 활용 및 체커보드 패턴의 텍스처를 이용해 직접 이미지를 합성하는 방법을 선택했다. 이는 승용차에서 일반적으로 aroundview를 구성할 때 이용되는 방식이다.

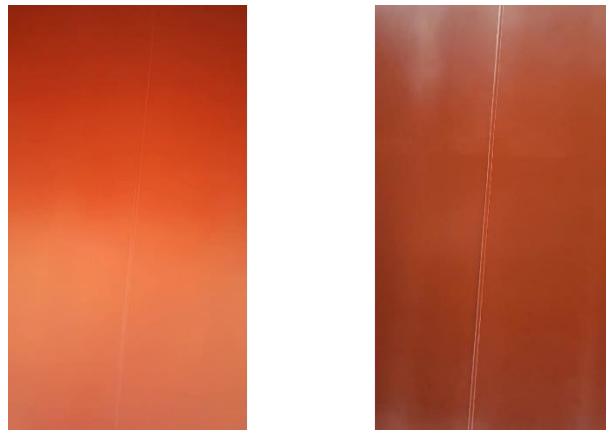


Figure 9. 도장면 이미지 예시

먼저 비스듬히 촬영된 이미지를 orthogonal한 이미지로 변환하기 위해 원근변환행렬은 구해야 한다. Unity상에서 체커보드 패턴의 텍스처를 적용한 Object를 촬영, 4개의 특징점을 지정한 후 특징점 사이의 비율을 이용해 원근 변환 행렬을 구한다.

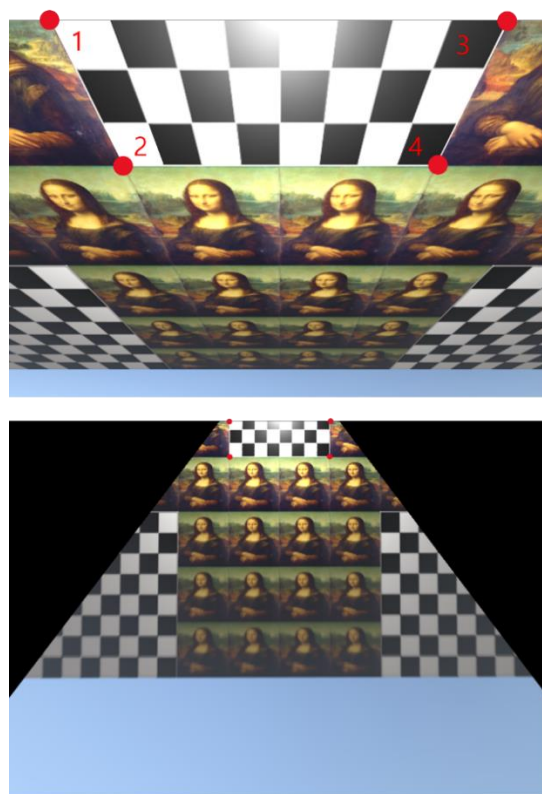


Figure 10. 원근변환 과정

촬영된 원본 이미지에서 4개 점의 픽셀 좌표를 구한다. 변환 후 작은 체커보드 한 칸의 크기를 임의로 설정한 후 4개 점의 좌표를 입력 시 openCV를 이용해 원근 행렬을 구할 수 있다.

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

원근변환행렬을 4대의 카메라로 수집한 영상에 적용 시 orthogonal한 aroundview 영상을 구현할 수 있다.

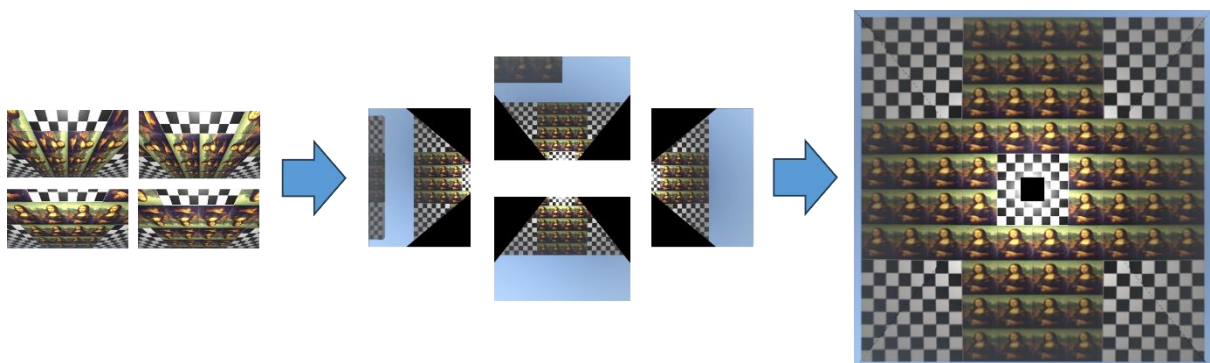


Figure 11. aroundview 구성 과정

---

## 3.2. YOLO

### 3.2.1. 학습 데이터 수집 및 데이터셋 구성

삼성중공업 거제조선소에 있는 실제 하부도장면 영상을 바탕으로 학습 데이터를 구성하였다. 영상은 장치가 움직이는 경로를 따라 촬영하였으며, 모든 상황에서 대응을 위해 용접선, 강재경계선, 장애물 등 다양한 정보를 수집할 수 있도록 하였다. 이 후 촬영한 영상을 프레임 단위로 추출해서 학습 이미지를 수집했다. 본 과정을 통해 얻어진 이미지는 약 3500장 정도로 모두 학습 데이터로 사용되었다.

### 3.2.2. 학습 데이터 라벨링

YOLO모델 학습을 위해서 이미지 라벨링이 필요하다. 데이터 셋을 만들기 위해 이미지에서 boundingbox를 지정하여 라벨링을 수행한다. boundingbox의 경우 정규화된 수치좌표로 나타내야 하며, 이를 위해 이미지 라벨링 툴인 laellmg를 이용하여 이미지 라벨링을 수행하였다. 라벨링은 총 6가지 클래스를 나누어 라벨링을 했으며 종류는 다음과 같다.

0. welding line – 용접선 : 작업 시 기준선의 역할을 할 object
1. paint border – 페인트 경계선 : 작업 시 도장 경계 역할을 할 object
2. carrier border – 지지대 경계선 : 작업 시 기준선의 역할을 할 object
3. obstacle – 장애물 : 도장 불가 지점
4. plate border – 도장면 경계선 : 진입점의 기준이 될 object
5. hole – 구멍 : 도장 불가 지점

### 3.2.3. YOLO 학습 결과

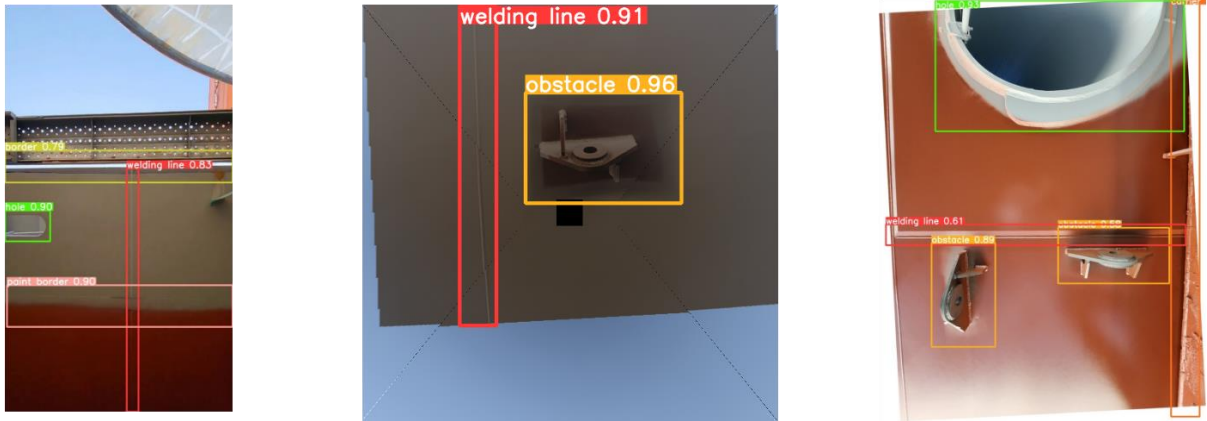


Figure 12. YOLO 학습 결과

### 3.2.4. YOLO 학습데이터 증강

객체 탐지를 할 때 bounding box가 예상 외로 잘 잡히지 않는 문제가 있다. 이 문제의 원인을 턱없이 부족한 데이터셋이 부족하다고 판단하였으나, 새로운 데이터 수집이 어려워, 기존 데이터를 기반으로 데이터를 증강하였다. 데이터 증강을 통해 모델 성능을 향상시켜 bounding box가 더 잘 잡히는 것을 목표로 하고 진행했다. 대부분 이미지 좌우 반전 및 명암 조절을 통해 데이터를 증강하였고, 데이터 증강 이후 학습 시 다음과 같은 모델 성능 지표 개선이 보여졌다. 경험적으로도 데이터 증강 이후 객체가 더 잘 잡히는 모습을 볼 수 있었다.

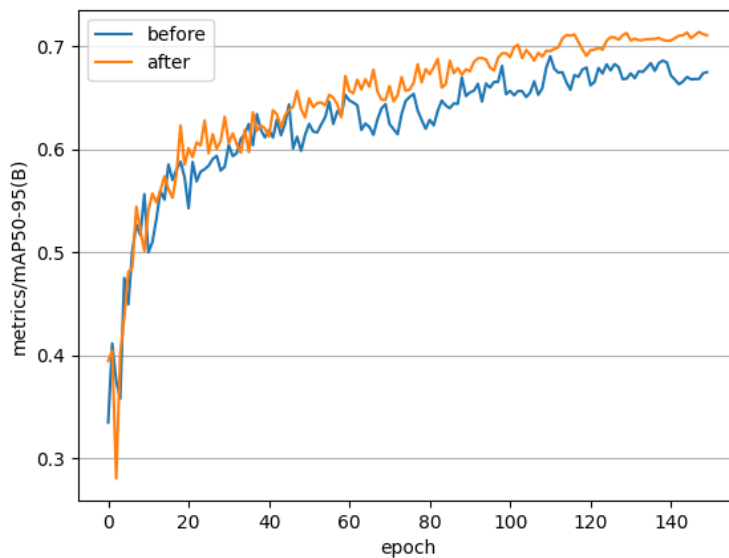


Figure 13. 데이터 증강 결과



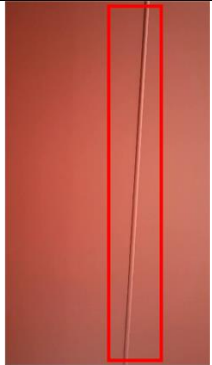




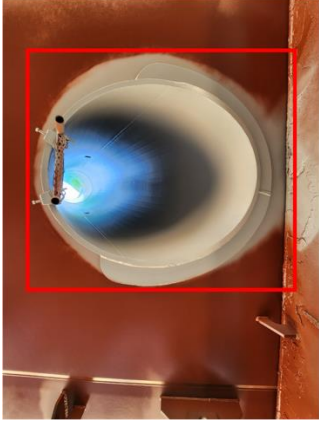
		
Welding line	Paint border	Carrier border
		
Obstacle	Plate border	hole

Figure 14. Class별 예시



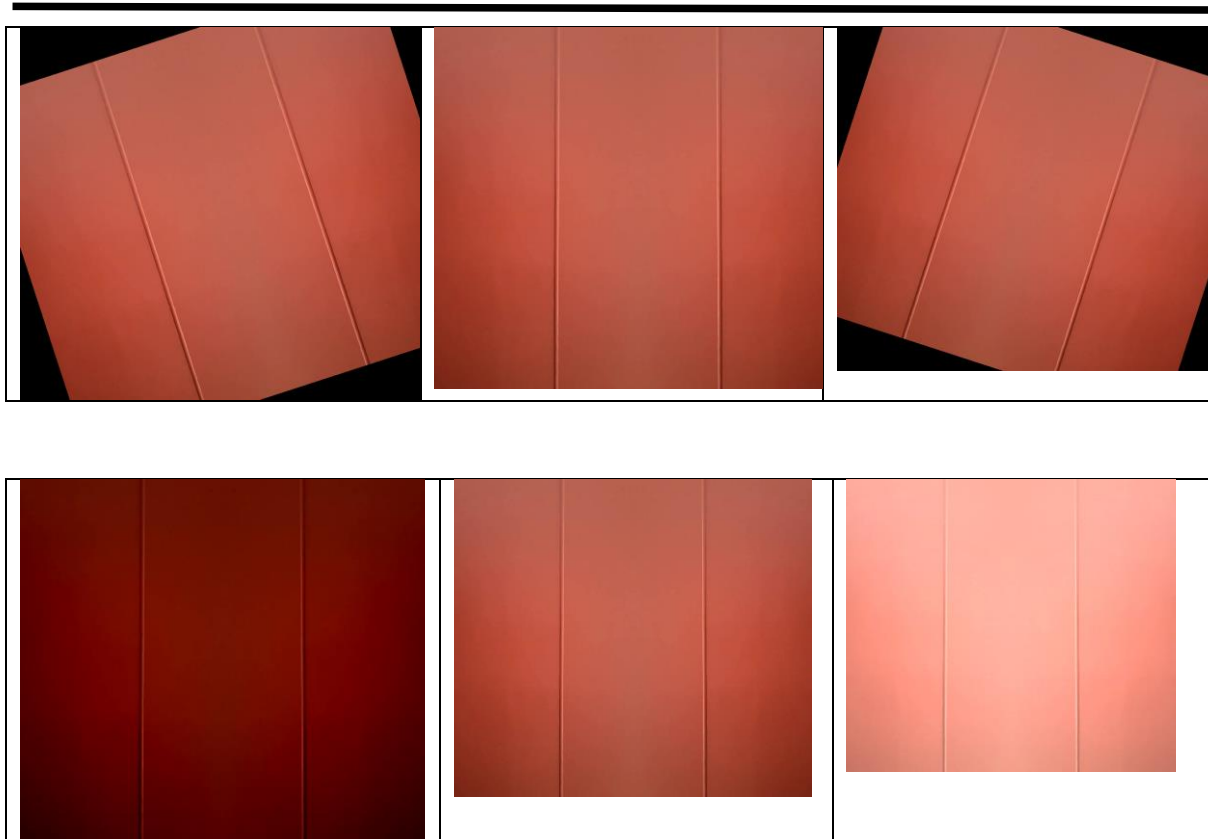


Figure 15. 증강 예시

### 3.3. line detecting

도장 기준선 검출을 위해 YOLO모델을 이용한 객체 검출 결과를 이용하였다. Airline을 이용해 bounding box 내부의 선을 검출하고 검출된 선분이 지나는 점들을 활용해 curvefit를 적용, 하나의 직선을 검출한다.

bounding box 내부에 Airline을 적용한 결과는 아래 그림과 같다. Airline은 끊어지지 않는 긴 선의 표현 및 경향이 같은 선을 병합하여 나타냄을 표방하나, 원본 데이터의 용접선이 너무 희미하거나 두꺼워 예상보다 많은 선이 검출되었다. 또한 도장면의 색수차가 적고, bounding box에 근접하는 장애물이나 테이프 등으로 인해 필요하지 않은 선들이 검출되었다. 그러나 본 과제의 요구조건은 용접선, 강재 경계선 등과 가장 유사한 하나의 기준선이므로, 선분들을 하나의 선으로 표현 할 필요가 있다.

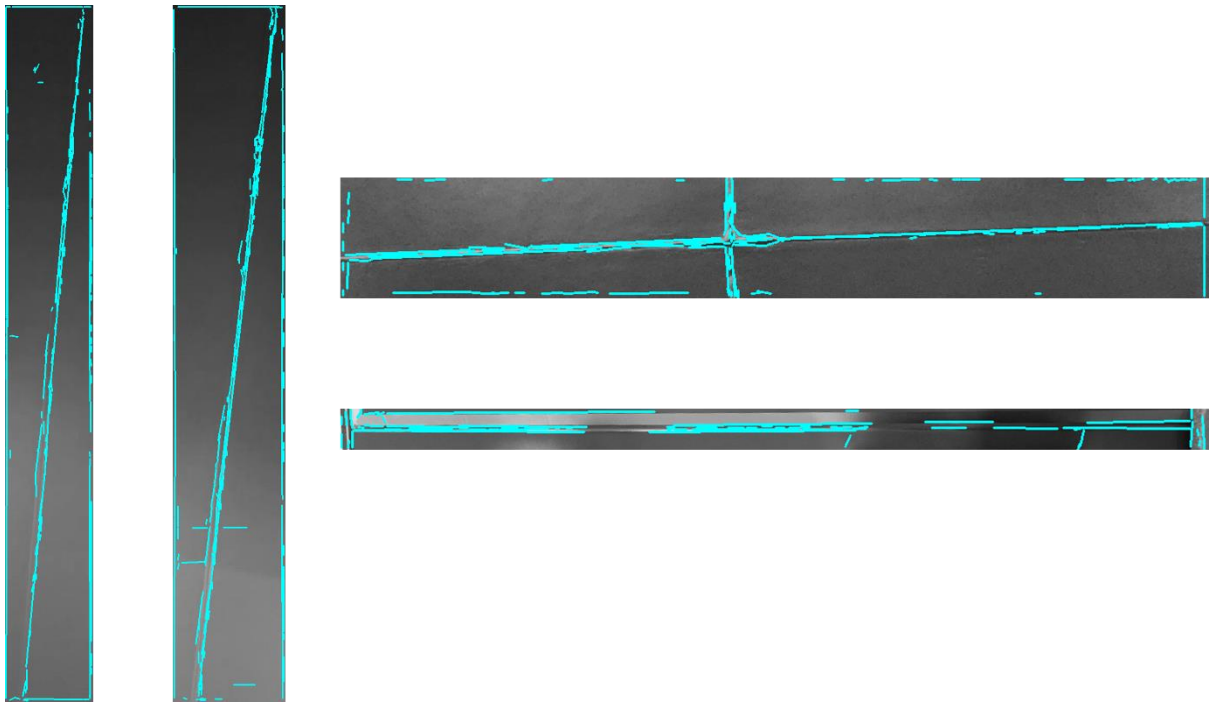
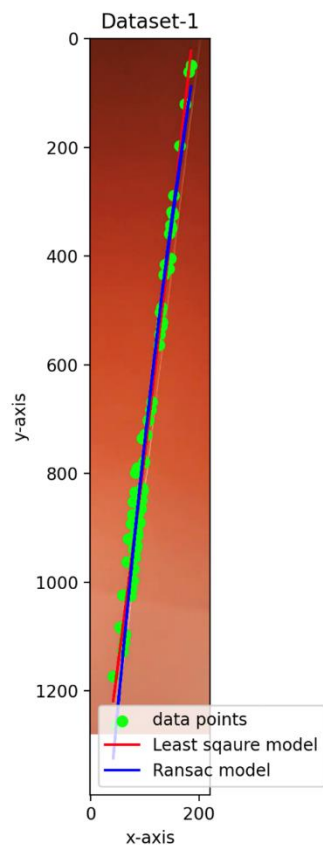


Figure 16. Airline line detection 결과

최초에 Airline으로 검출된 직선들의 기울기 평균을 이용해 하나의 선으로 표현해 보았으나, 일부 상황에서 선이 제대로 표현되지 않았으며 디버깅 결과 outlier에 많은 영향을 받고 있음을 알 수 있었다. 절사평균, 중앙값 등의 지표를 이용하여 outlier를 배제하려 시도했으나, 성과가 미미하였다. 따라서 line fitting 방법을 활용해 보았으며, hough 변환, Ransac알고리즘, curvefit 세가지 방법을 검토 및 적용해 보았다. hough 변환 및 Ransac은 이전 방법보다 선을 잘 검출하였으나, 설정해야 할 파라미터가 많아 상황에 따른 변화에 유동적으로 대처할 수 없었으며, hough의 경우 실시간성 또한 저해되었다. 직선의 방정식과 Airline으로 주어지는 이산적인 좌표 데이터를 파라미터로 사용하는 curvefit를 사용하였다.



Ransac



Curve\_fit

Figure 17. line detection 테스트

curvefit를 사용하여 선이 보다 정확하게 검출되었으나, 여전히 outlier에 영향을 받고 있어 선이 틀어지는 경우가 발생했다. Airline에서 적절히 검출된 선의 경우 비교적 길이가 길었으며, 이를 바탕으로 너무 길이가 짧은 선은 outlier로 규정하여 제외했다. 또한 검출하여야 할 선은 bounding box의 중심을 가깝게 지나므로 직선의 방정식을 구했을 때 bounding box의 중심에서 멀리 벗어나는 선들 또한 outlier로 규정하여 제외하였다.

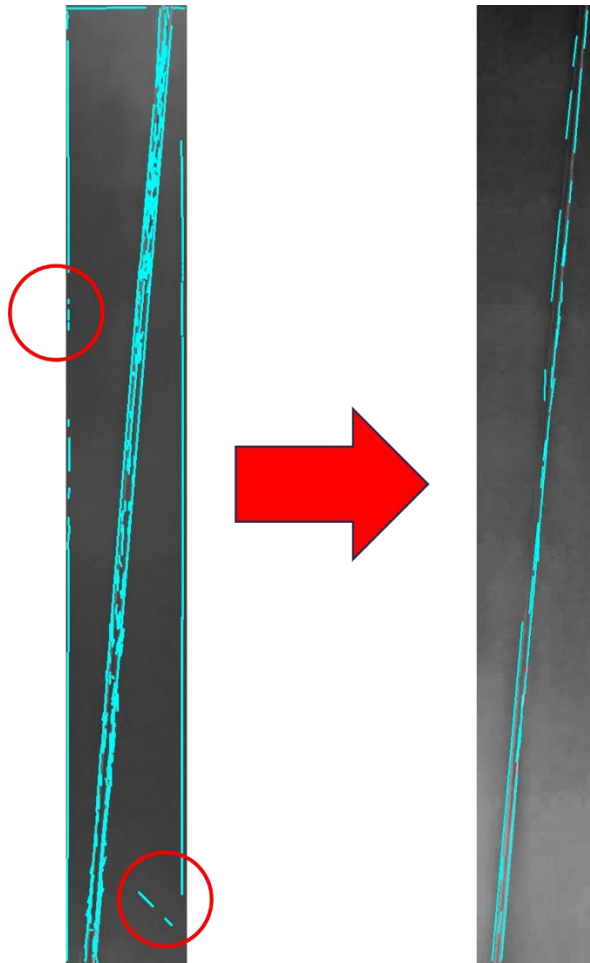
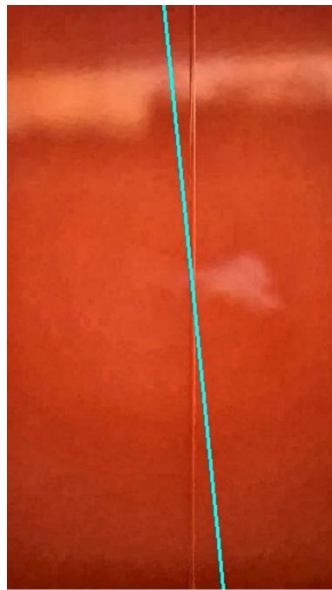
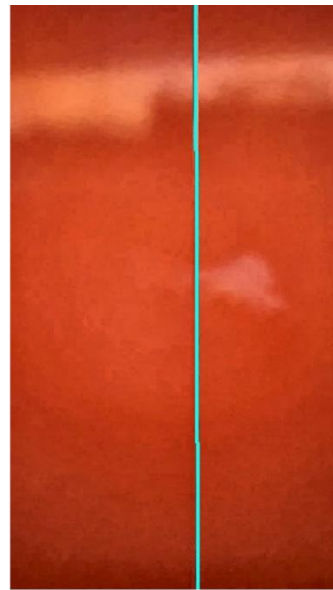


Figure 18. outlier 제거

outlier 제외 후 선의 흔들림이 감소하였으며 대부분의 이미지에서 선이 잘 검출되었다. 그러나 간헐적으로 검출되는 선분이 실제 검출되어야 할 선분과 동떨어진 곳에서 검출되는 경우가 존재하였다. curvefit로 직선을 구할 때 최소제곱 회귀분석법을 사용하는데, 초기값의 설정 즉 이전 프레임의 기울기 정보 또한 중요하며, 잘못된 결과로 수렴될 가능성이 존재한다. 따라서 절사평균을 활용하여 편차가 큰 기울기 값의 경우 outlier로 규정하여 제외하였다. 기울기의 절사평균을 활용해 curvefit의 결과 기울기 값을 제한하여 기준선이 튀지 않고 연속적으로 나타나게 할 수 있었다.



Bound 설정 X



Bound 설정 O

Figure 19. Boundary 설정을 통한 noise 제거

검출된 기준선을 보다 쉽게 인식할 수 있도록 시각화 하였다. 또한 각도 및 기준선과 중심간의 거리, 선분의 좌표등의 정보를 딕셔너리 형태로 저장하여 이미지정보를 Unity로 전송할 수 있도록 하였다.



Figure 20. line detection 영상 적용

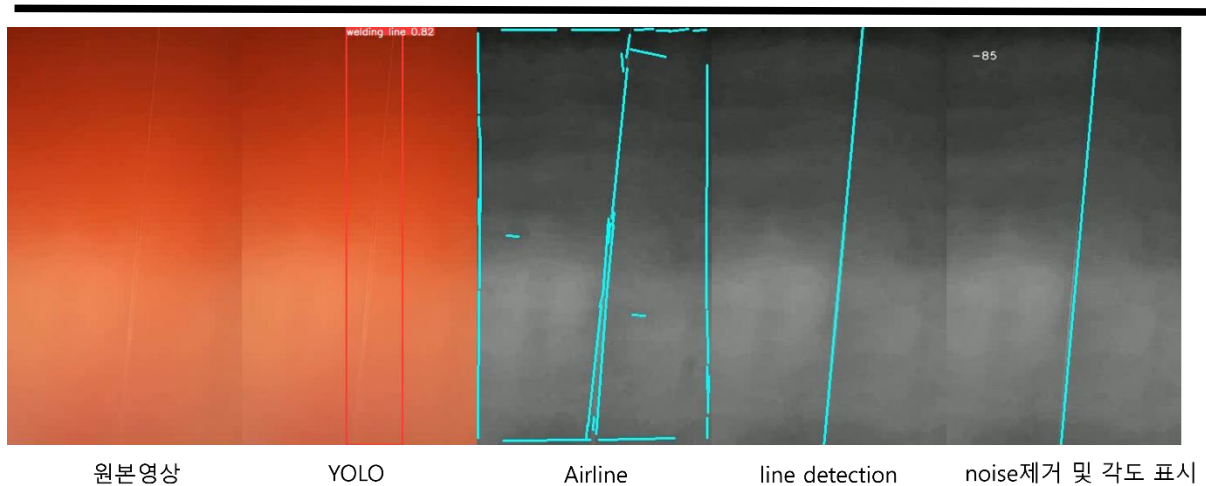


Figure 21. line detecting 적용 과정

### 3.3.1. Unity

앞서 구현한 내용의 시각화 및 제어 신호를 이용한 자율주행을 효과적으로 나타내기 위하여 Unity를 기반으로 한 시뮬레이터를 구현하였다. Unity 멀티스레딩 및 소켓 통신을 통하여 파이썬과 Unity(C#)간의 통신이 이루어지며, 과정은 다음과 같다. 먼저, aroundview구성 및 도장면 정보 파악을 위해 4개의 영상을 어안렌즈로 촬영, 해당 영상정보를 파이썬으로 전송한다. 파이썬에서는 전송받은 영상정보를 활용하여 이미지 프로세싱을 하여 해당 정보를 다시 Unity(C#)로 보낸다. Unity에서는 전송받은 영상 정보를 이용해 영상 스트리밍 및 제어 신호를 생성하여 도장 로봇을 움직인다.

도장 로봇의 경우, 실제 차량과 동일한 움직임을 위해 휠 콜라이더를 활용하여 1개 조향장치와 4개의 바퀴를 이용하여 주행한다. 휠 콜라이더에 토크를 적용하여 차량의 수직방향 움직임을 제어하고 스티어링 휠을 적용하여 좌/우 회전을 구현, 차량의 핸들에 해당하는 움직임을 구현한다.



Figure 22. 휠콜라이더 적용된 도장로봇

도장 로봇의 움직임은, 상태 및 제어신호에 따라 달라진다. 도장로봇의 주행은 진입점 및 기준선에 따르며 진입점은 주로 'plate border'가 되고 기준선은 검출된 선들 중 가장 가까운

선으로 한다. 최초 도장로봇의 상태는 'BeforeEntering' 이며 소켓연결이 수립되고 진입점의 역할을 할 'plate border'가 파악될 시 차량이 움직이기 시작한다. 차량이 진입점을 지나는 순간 'Passing'상태가 되고, 진입점을 지나고 일정거리만큼 전진하면 'Painting' 상태가 된다. 'Passing'및 'Painting'상태일 때 기준선과 일정거리를 유지하며 직진한다. 이 때 일정거리 이상 벗어날 경우, 비스듬히 움직여 기준선과 일정 거리를 유지한다. 또한 기준선의 각도를 이용하여 직진상태를 유지한다. 'Painting' 중 진입점 탐지 및, 진입점과의 거리가 일정 이하가 되면 다시 'Passing'상태가 되며, 'Passing' 이후 'Rentry'상태가 되어 U턴하여 다시 도장작업을 계속한다. 1회 왕복 시 'JobFinish'상태가 되고 일련의 과정이 종료된다.

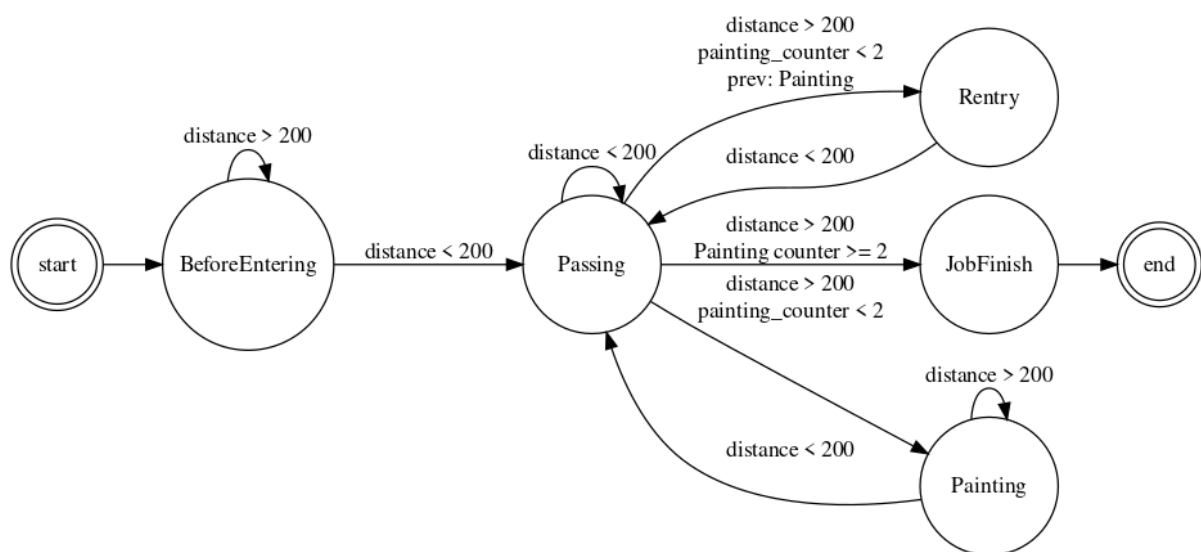


Figure 23. 도장로봇 제어 유한 상태 머신

## 4. 연구 결과 분석 및 평가

### 4.1. 설계 변경 내역

도장 환경을 소형화 하여 현실세계에 적용 가능함을 보여주기 위하여 라즈베리 파이와 차량 모듈, 멀티카메라 모듈, OV2640 기반 어안렌즈 카메라등을 활용하여 테스트 베드를 구축하고자 하였으나, 재료비가 부족하여 값싼 모듈을 사용하다 보니 멀티카메라 모듈 등에서 원인을 알 수 없는 오류가 발생하여, Unity를 이용하여 시뮬레이터를 구성, 시연을하기로 결정하였다.

### 4.2. 연구 결과

Figure 24를 통해 시뮬레이터의 가동장면을 볼 수 있다. 먼저 시뮬레이터에서 도장 로봇 상

하좌우의 영상 촬영 및 영상정보를 이미지 처리를 위해 파이썬으로 보내면, 파이썬에서는 영상을 읽어와 이미지 calibration/stitching 등의 과정을 거쳐 aroundview 이미지를 구성, 해당 이미지를 이용해 YOLO모델로 클래스를 구분, Airline으로 line을 detecting 하여 curvefit를 이용해 하나의 깔끔한 line을 detecting한다. 또한 이미지 프로세싱 결과 정보를 다시 Unity 시뮬레이터로 전달, Unity에서는 해당 정보를 활용하여 제어 신호를 생성하고 제어 신호에 따라 경로에 맞추어 도장로봇이 주행한다. 주행 시 차량과 기준선 간의 거리 및 각도를 맞추어 주행하며, 외부 요인으로 인해 경로를 벗어날 시 복원하여 주행한다.

시뮬레이터 실행 시 Start 버튼을 누를 시 통신 및 시나리오가 시작되고, 1회 왕복한다. 이 때 주행상태를 하단에 출력한다. 왕복 도장 작업이 끝날 시 시나리오가 종료된다.



Figure 24. Unity 기반 시뮬레이터 동작 화면

## 5. 결론 및 향후 연구 방향

### 5.1. 결론

이상으로 자율주행 도장로봇을 구현해 보았다. 본 과제는 RGB 카메라와 딥러닝 모델만을 활용하여 한정적인 상황에서 도장 자동화의 가능성을 시사하며, 적절한 물리엔진을 사용한 시뮬레이터로 실제 도장환경에서의 적용 가능성을 제시하는 바이다. 해당 과제를 실제 산업 현장에 적용 시 작업자 안정성 향상, 작업 효율성 및 생산성 증대를 기대할 수 있다. 또한 본



과제를 기반으로 하여 추가적인 이미지 또는 작업로그 등의 데이터를 수집한다면 보다 안정적인 도장 자동화 모듈을 구현할 수 있을 것으로 보인다.

## 5.2. 향후연구

### 5.2.1. 실제 도장환경 테스트 베드 구축

Unity 및 Unity 기반 물리엔진을 활용하여 실제와 유사한 도장환경 및 도장로봇을 구현하였으나, 여전히 현실의 도장작업과 괴리는 존재한다. 또한 실제 도장환경과 일치하는 테스트 베드를 구축 시, 부착된 카메라를 활용하여 모델학습을 위한 지속적인 영상 수집이 가능할 것으로 예상된다. 그 밖에 실제 환경에서만 얻을 수 있는 다양한 도장 시나리오를 취득할 수 있어 보다 정교한 도장 자동화 모듈을 만들 수 있을 것으로 예상된다.

### 5.2.2. 작업경로 생성 모듈 및 작업차량 위치 추적 프레임워크 개발

제어 신호를 이용한 도장 작업화에서 더 나아가, 광범위한 영상 데이터 수집 후 작업경로 생성 및 실시간 작업차량 위치 추적 프레임워크 개발하여 유사 시 작업차량 원격제어 및 안전 사고 예방을 기대할 수 있다. 또한 경로 생성을 통해 자동화의 수준이 증대함에 따라 인적 오류를 줄여 일정한 품질 및 속도를 보장하여 생산성을 높일 수 있을 것으로 예상된다.

## 6. 구성원별 역할 및 개발 일정

### 6.1. 개발일정

5월		6월					7월				8월					9월				10월	
4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주
관련 기술 지식 학습																					
		데이터 수집 및 전처리																			
		YOLO 모델 학습																			
				around view 구현																	
				Edge detection 모델 적용																	
					직선 검출 알고리즘 적용																
							중간 보고서 작성														
											Unity 시뮬레이션 구현										
											Edge detection 성능 향상										
											YOLO 학습 데이터 증강										



---

and Electrical Engineering (COSITE), 2023 2nd International Conference on. ,pp 260-265, 2023

[1] Juan R. Terven, Diana M. Cordova-Esparza, arXiv, A Comprehensive Review of YOLO: From YOLOv1 and Beyond [Online]. Available: <https://arxiv.org/abs/2304.00501>

[2] Xiao Lin, Chen Wang, arXiv, AirLine: Efficient Learnable Line Detection with Local Edge Voting [Online]. Available: <https://arxiv.org/abs/2303.16500>

[3] OpenCV [Online]. Available: <https://docs.opencv.org/4.x/>

[4] Github, surround view system introduction [Online]. Available: <https://github.com/neozhaoliang/surround-view-system-introduction>

[5] Github, Airline [Online]. Available: <https://github.com/sair-lab/AirLine>

[6] Github, DexiNed [Online]. Available: <https://github.com/xavysp/DexiNed>

[7] Github, ultralytics [Online]. Available: <https://github.com/ultralytics/ultralytics>