제 2회 자율주행 모형차량 경진대회

Autonomous 최종발표

백재민 | 박준아

Auto
nomous

Autonomous 최종발표

Index

01 차량 공학적 설계

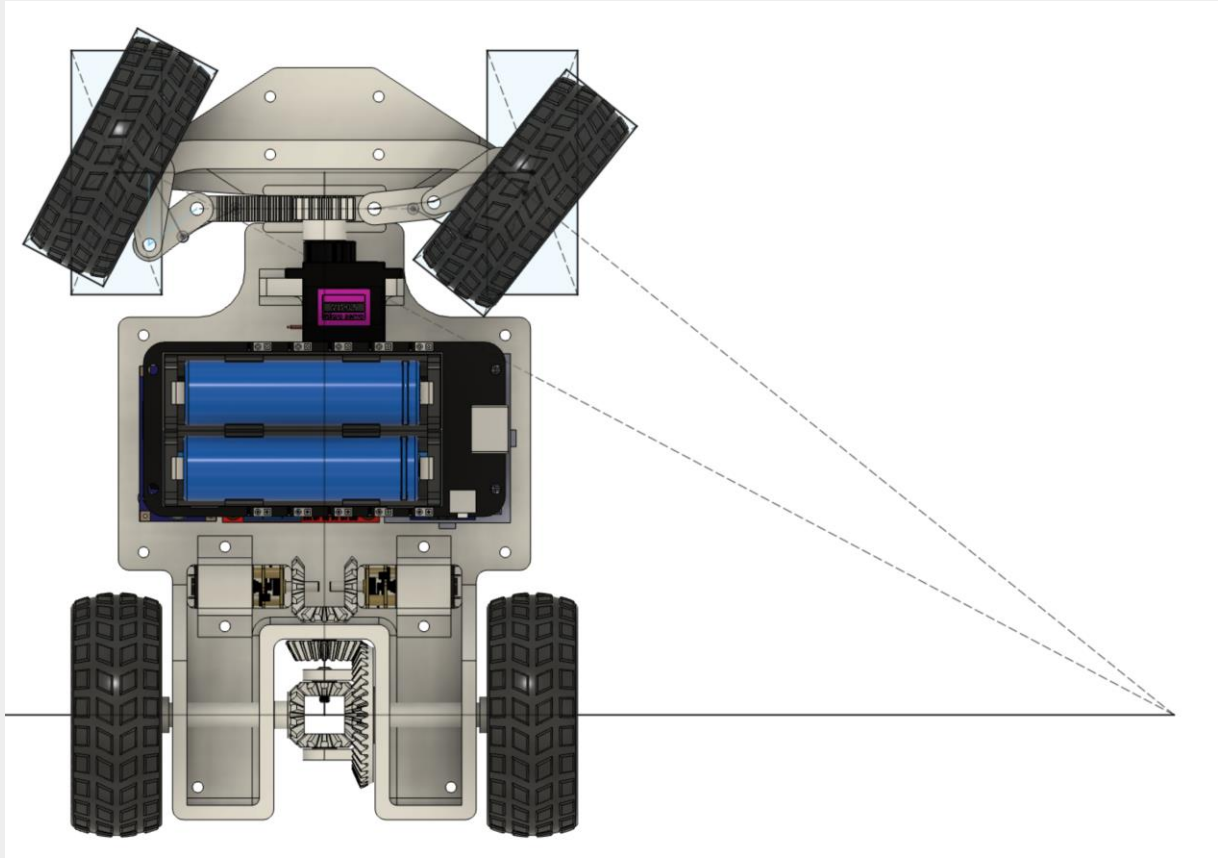
전륜 부분에서는 에커만 조향(Ackermann Steering) 을 사용하고,
후륜에서는 차동기어(differential gear) 설계를 적용하여 곡선 주행 시 슬립(slip) 현상을 최소화

전륜 : 에커만 조향

회전 중심을 향하는 방향으로 바퀴가
수직으로 정렬되도록 한다.

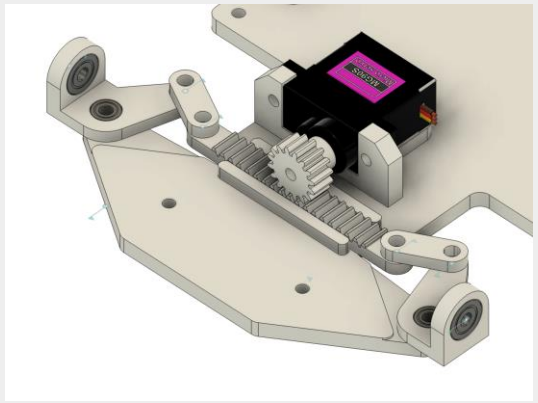
후륜 : 차동기어

곡선 주행 시 외측 바퀴는 빠르게,
내측 바퀴는 더 느리게 회전한다.



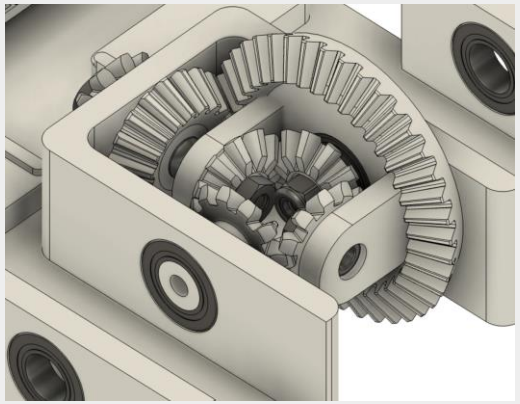
01 차량 공학적 설계

전륜 부분에서는 에커만 조향(Ackermann Steering) 을 사용하고,
후륜에서는 차동기어(differential gear) 설계를 적용하여 곡선 주행 시 슬립(slip) 현상을 최소화



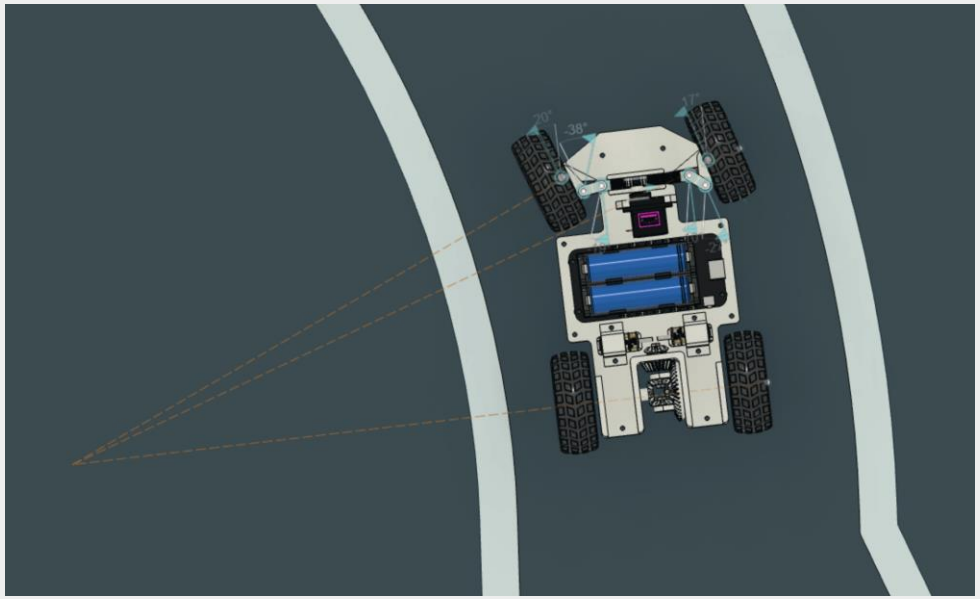
에커만 조향

+



차동기어

→

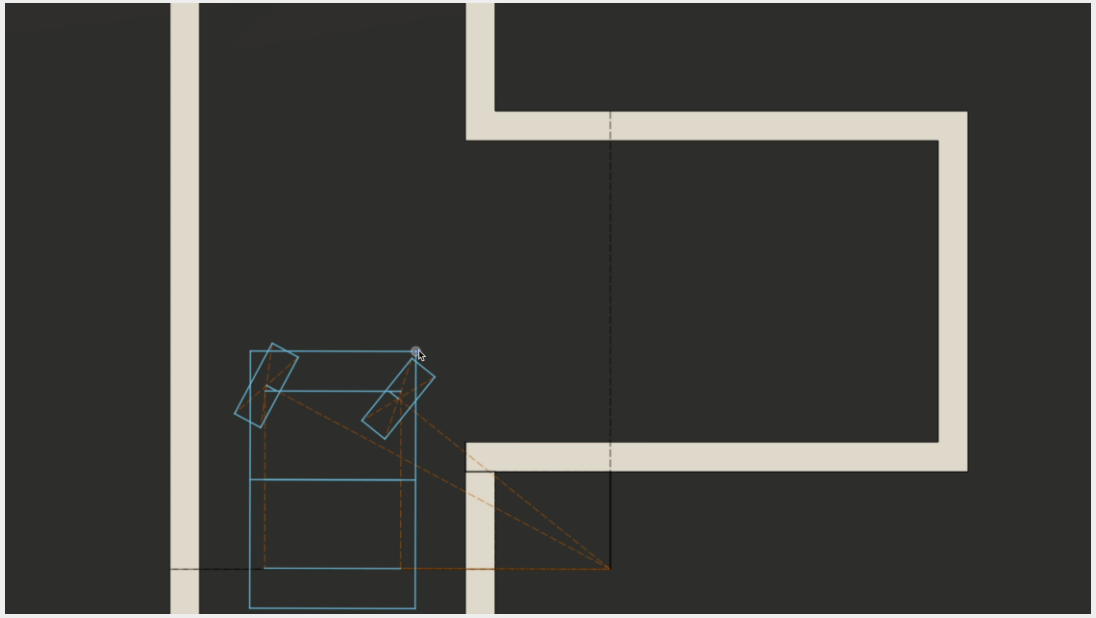


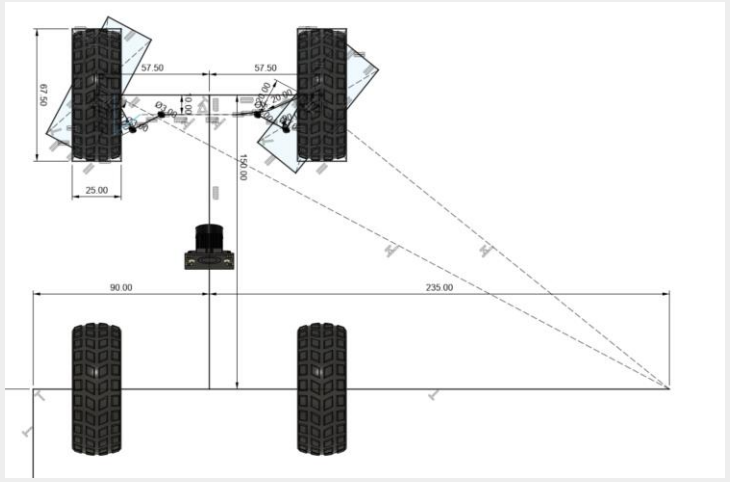
에커만 조향과 차동기어를 직접 3D 모델링하여 경진대회에
적합한 크기로 제작

주행트랙을 3D모델링 하고 주행을 시뮬레이션 해봤을 때
바퀴가 최적의 접지력과 회전력을 유지하도록 하여 운전의
안정성을 높일 수 있었다.

01 차량 공학적 설계

에커만 조향(Ackermann Steering) – 주차





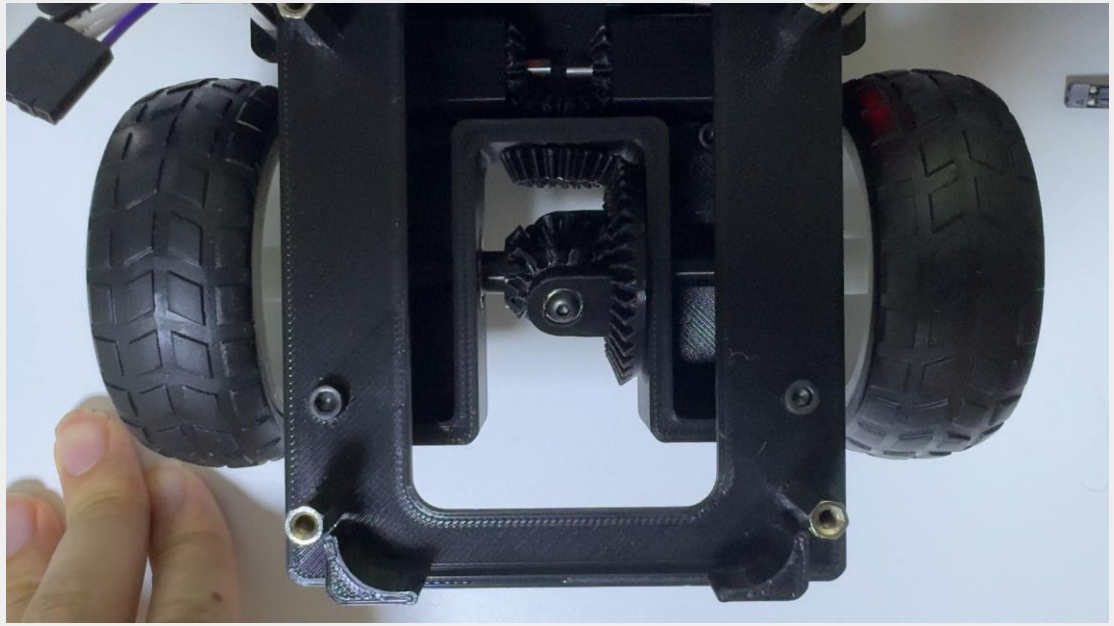
에커만 조향

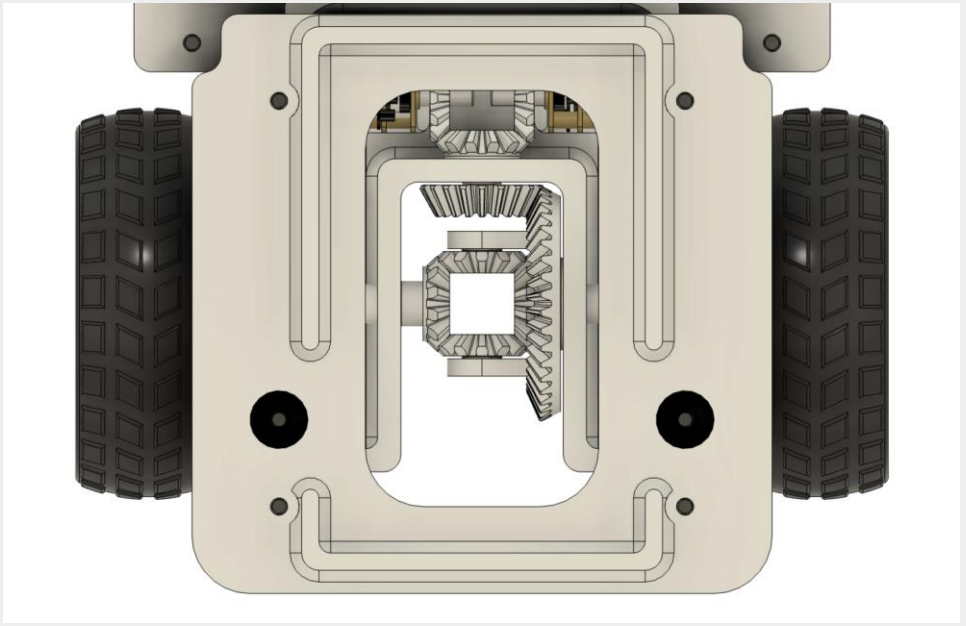
주차부분의 조향을 원활하게 하기 위해 235mm점을 기준으로 설계를 진행

영상 : https://drive.google.com/file/d/1LY4Bslq5zAYjF-1-Lb-CiBgC1ZaFgqJb/view?usp=sharing

01 차량 공학적 설계

차동기어(differential gear)





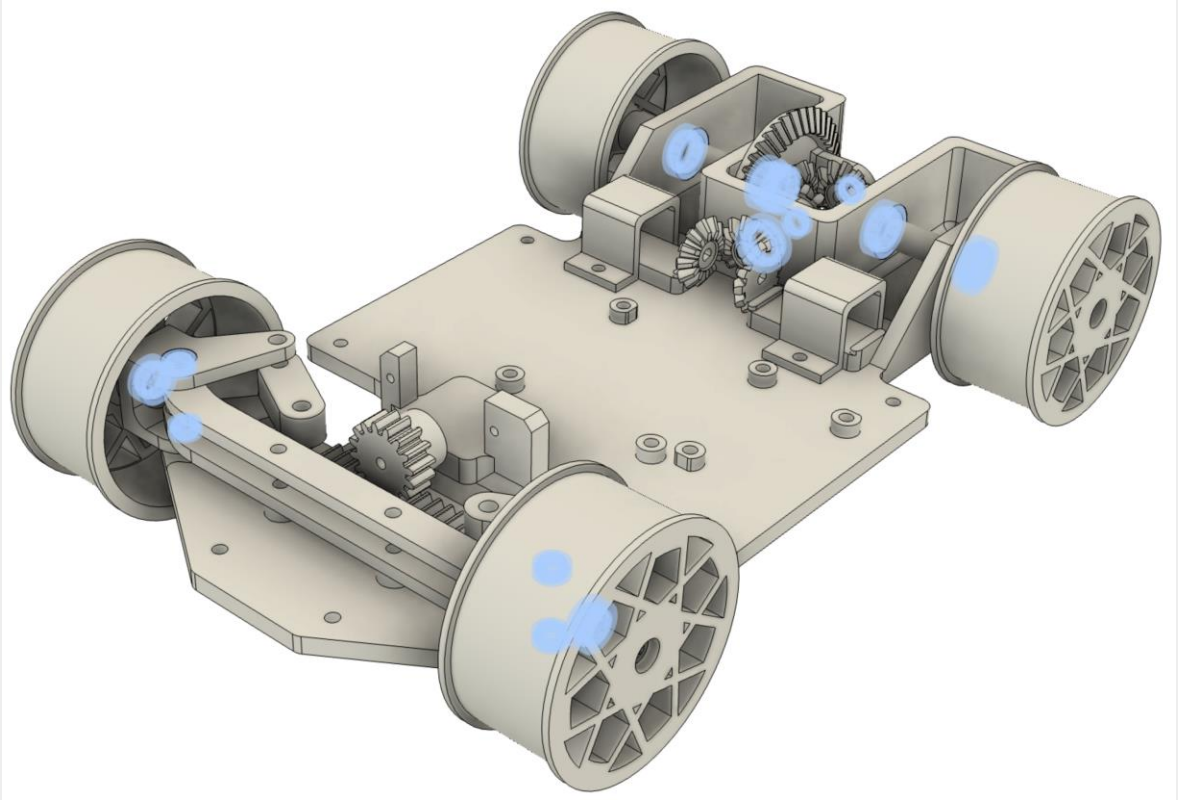
차동기어

차동기어의 베벨기어를 오차를 고려하여 직접 설계하고 베어링을 활용하여 마찰을 최소화, 그 결과 곡선에서의 주행이 훨씬 부드럽게 주행 가능

영상 : https://drive.google.com/file/d/1AfJ19zceW9eVdtmJk0rsd53HG0S3NOwt/view?usp=drive_link

01 베어링 설계

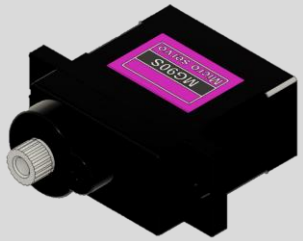
베어링을 활용하여 축 부분을 견고하고 훨씬 부드럽게 회전할 수 있도록 설계



14개의 베어링(파란색 부분)으로 바퀴의 구동부위의 안전성과 부드러움을 향상

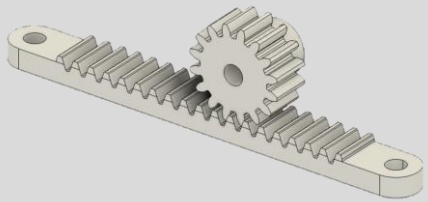
01 모터 설계

서보모터와 랙 앤 피니어 기어를 사용하여 조향, n20 모터 2개와 베벨기어를 사용하여 후륜구동



서보모터(MG90S) + 랙 앤 피니언 기어

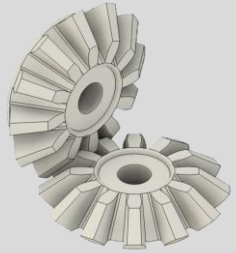
전자적으로 제어되어 정확한 각도로 회전



회전 운동을 직선 운동으로 변환

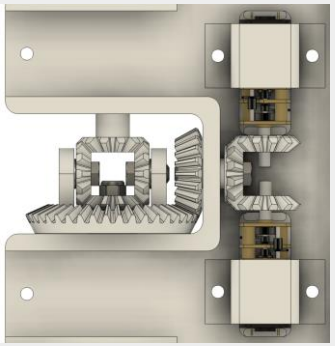
n20모터(300RPM) + 베벨 기어

공간제약으로 인해 소형이지만 충분한 토크를 낼 수 있음



두 축이 서로 직각으로 교차하는 경우에 회전력을 전달하는 데 사용되는 기어

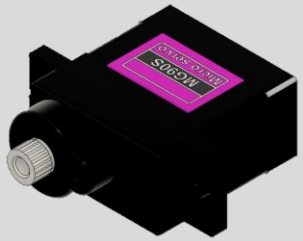
서보모터가 피니언을 돌리면, 랙이 좌우로 움직이게 되어 차량의 바퀴가 좌우로 회전하게 된다.



출력 후 테스트 결과 모터가 1개일때보다 2개를 이용할 때 더 큰 구동력을 가질 수 있었다.

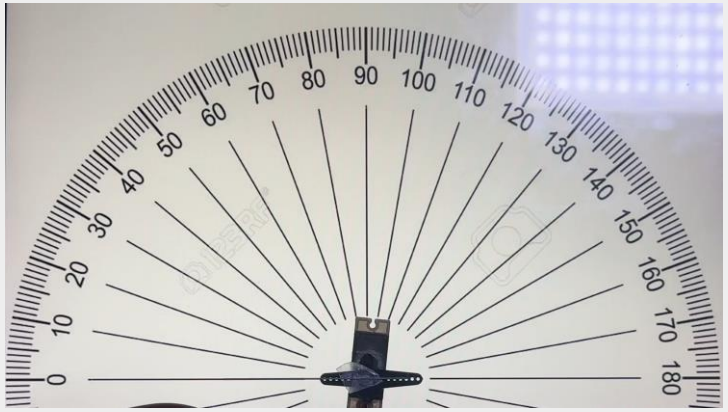
01 모터 설계

서보모터 각도 범위 설계

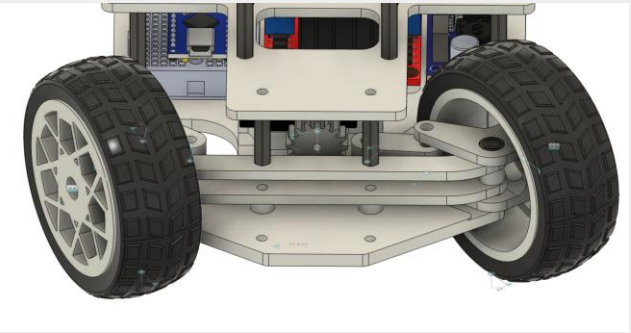


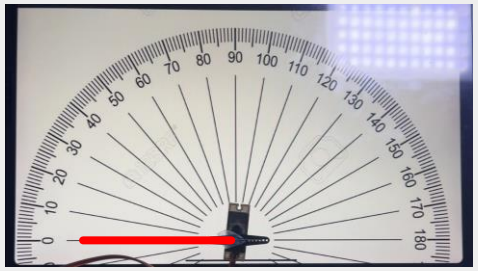
서보모터(MG90S)

전자적으로 제어되어 정확한 각도로 회전

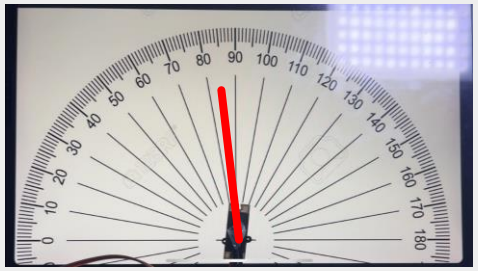


영상 : https://drive.google.com/file/d/1mKsQlWh7G1h_vtx871F1kROuQ4hWicur/view?usp=sharing

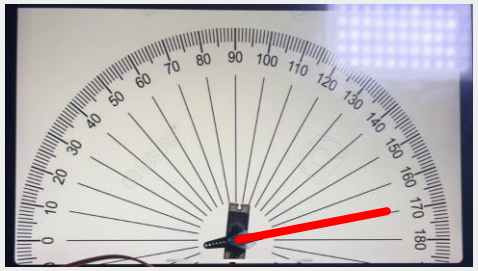




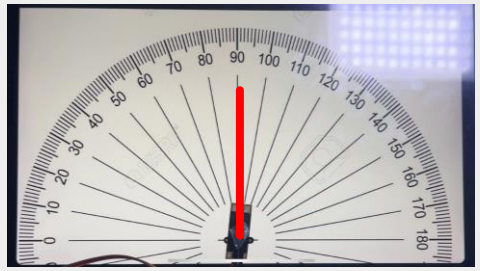
0°



90°



180°



95°

조향 각도의 범위를 10° ~170°로 설정하여 설계를 진행

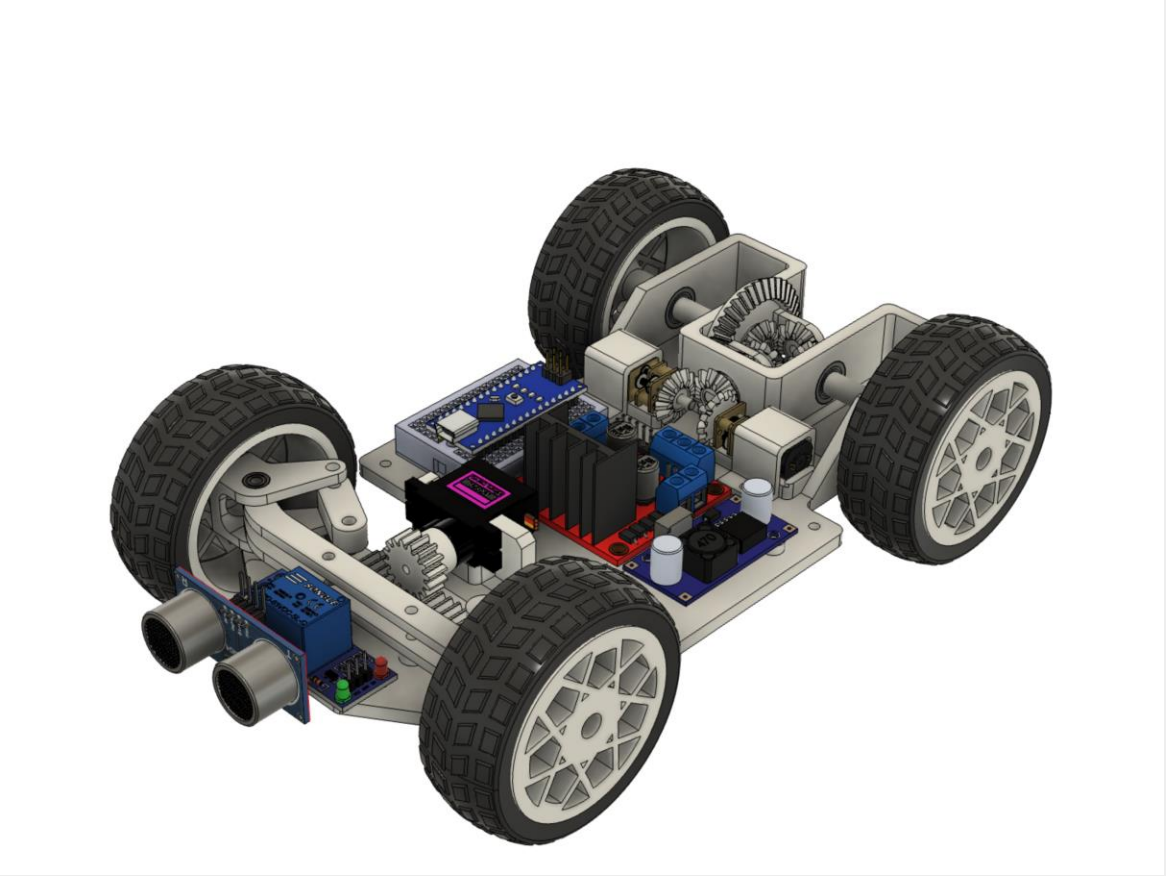
01 LED 설계

터널에서 쉽게 차량을 인식할 수 있도록 적절한 LED를 사용했다.

9개의 12V LED를 사용하고 전방 뿐만 아니라 측면으로도 LED를 배치하여 터널 속에서 훨씬 더 원할하게
차선을 인식할 수 있도록 설계

01 내부 설계

First Floor



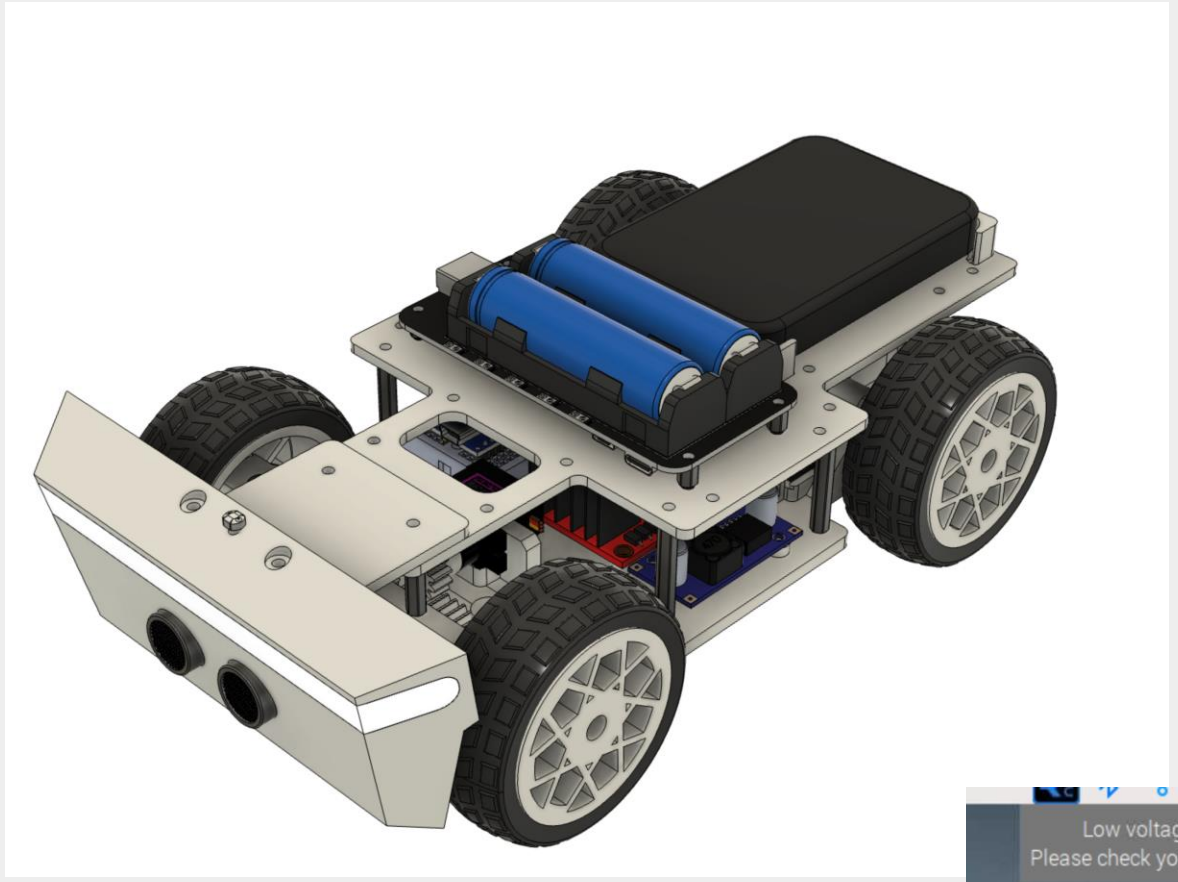
초음파센서, 릴레이모듈, 서보모터, 모터드라이버, n20모터, 아두이노

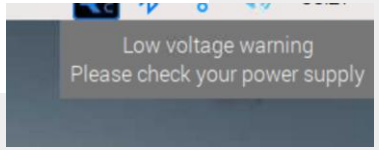
->선정리 용이

Auto
nomous

01 내부 설계

Second Floor



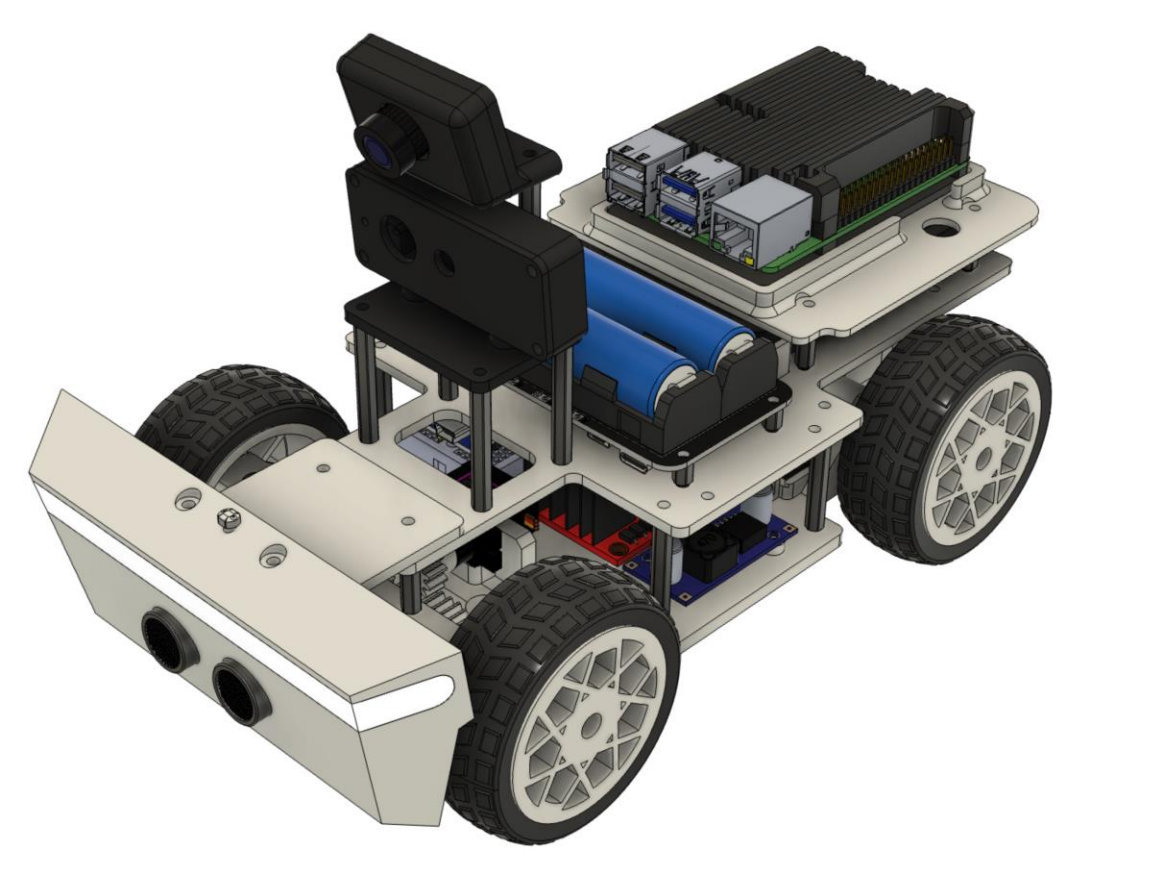


LED, 조도센서, 배터리

-> 적절한 전력을 위해 배터리는 2개를 사용

01 내부 설계

Third floor

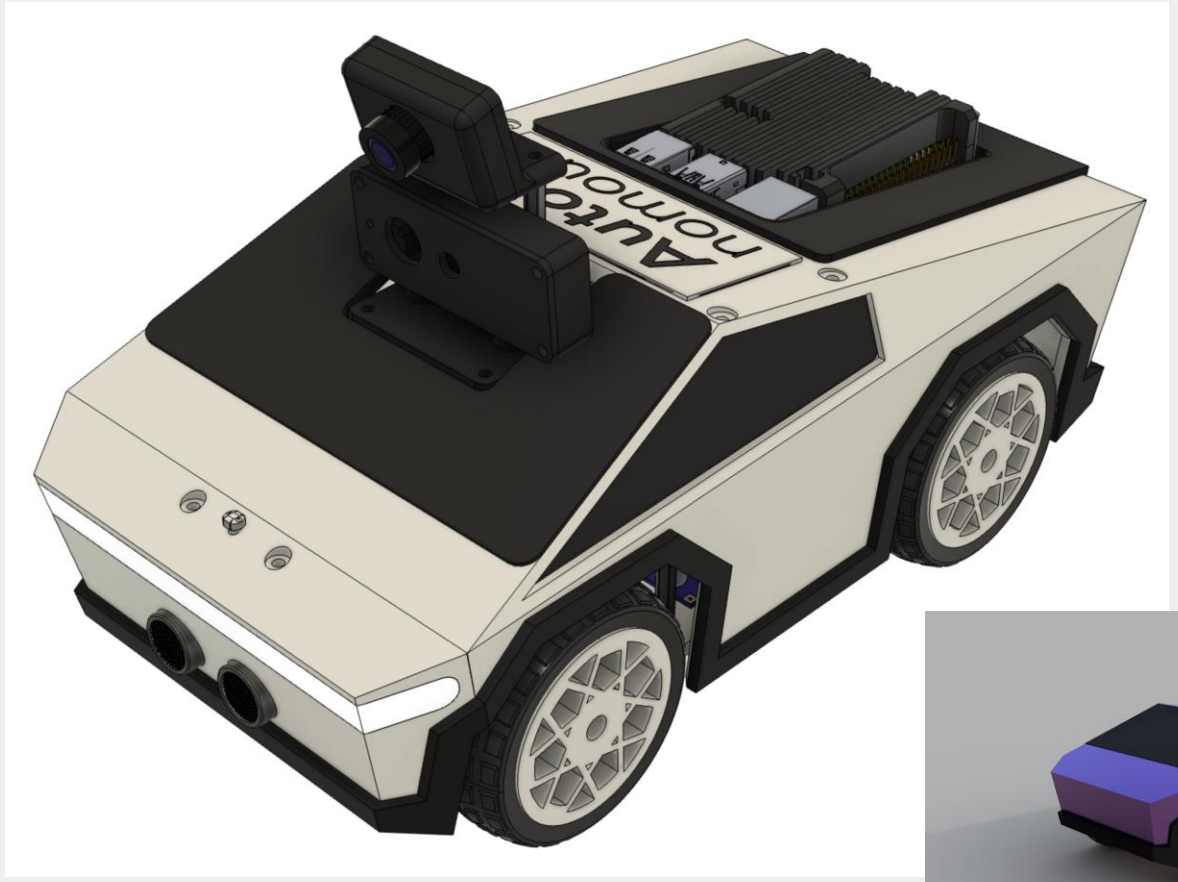


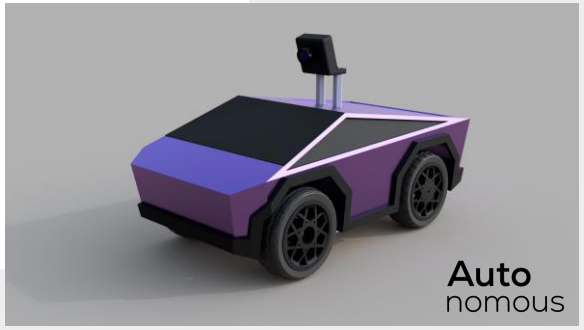
카메라. 라즈베리파이

->방열을 위해 라즈베리파이를 상단에 배치

01 내부 설계

Cyber Truck





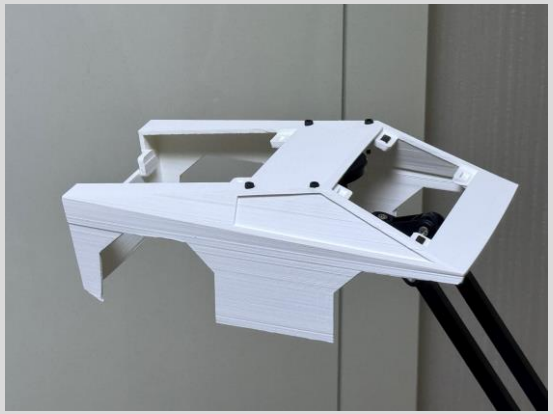
최종 외관 디자인은 사이버트럭을 오마주하여 설계

01 외관 설계

3가지 후보를 선정하여 직접 제작

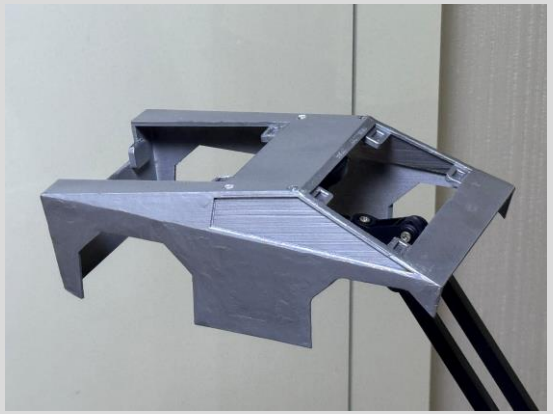
MASH

무게를 줄여 속도를 더 빠르게 할 수 있다는 장점이 있다.



SIMPLE

가장 간단한 모형으로 제일 적합하다고 판단했다.



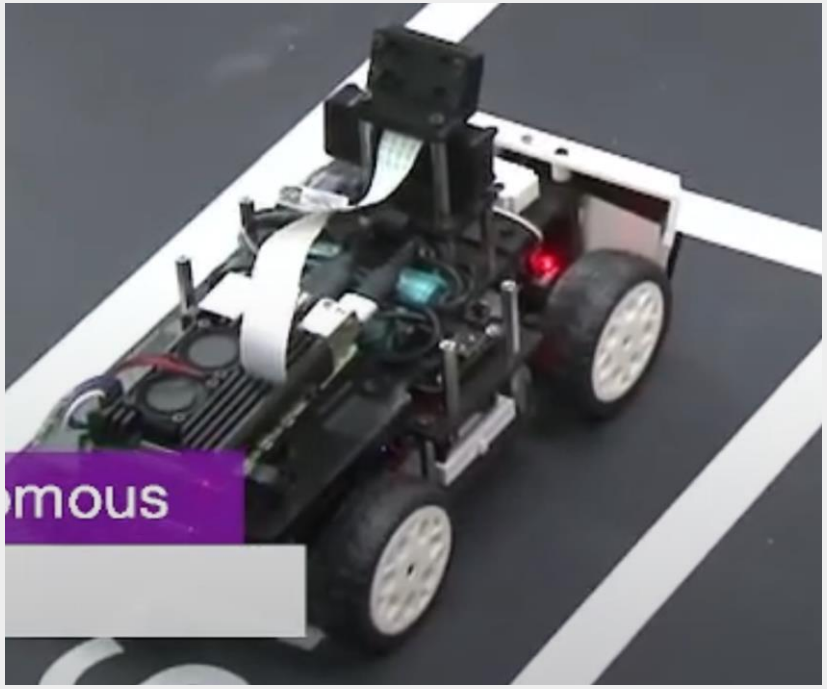
SILEVER

이쁘고 단단하지만 무겁다는 단점이 있다.

01 외관 설계

조금이라도 랩타임을 줄이기 위해 대회에서는 외관 탈거 진행

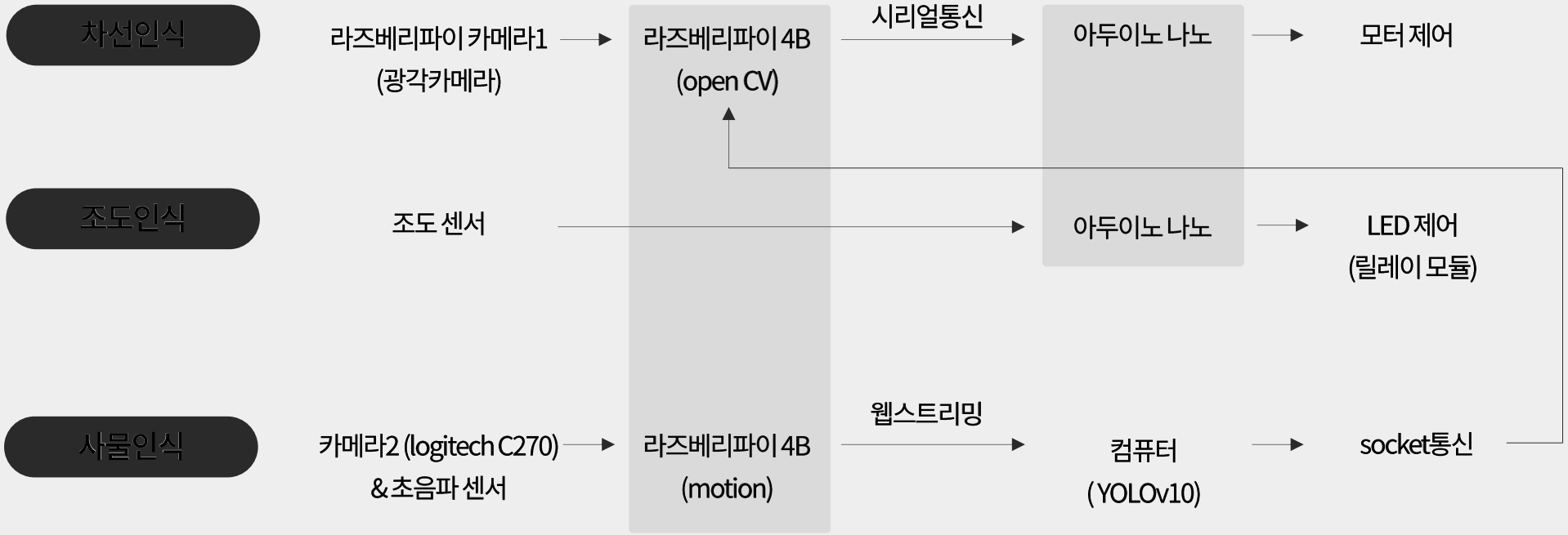
랩타임 : 약 24초
(직접 타이머 측정)



랩타임 : 약 22초
(직접 타이머 측정)

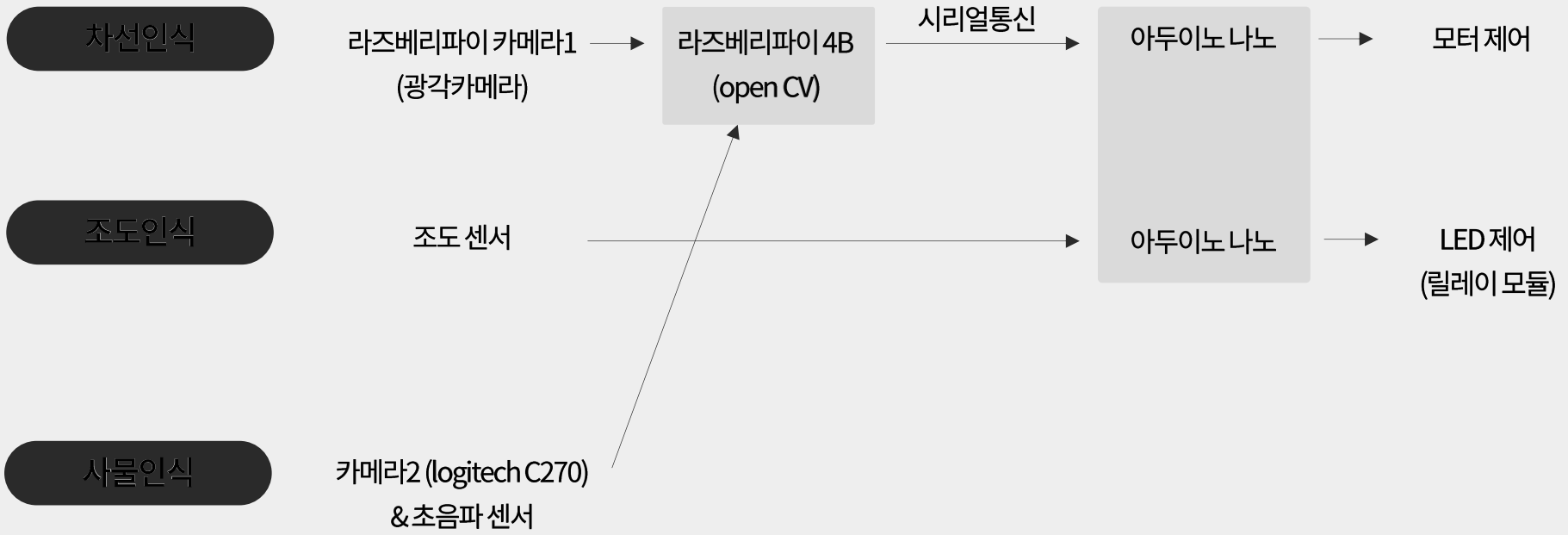
02 차량구동 – 인식 – CASE1

차량구동을 위해서는 차선인식과 사물인식이 필수적이다. 따라서 차선인식에 라즈베리파이와 아두이노 나노를 사용하고,
사물인식(신호등 색상포함)은 라즈베리파이를 사용하여 컴퓨터로 영상을 전송하고 이를 다시 라즈베리파이에 결과값을 전송



02 차량구동 – 인식 – CASE2

차량구동을 위해서는 차선인식과 사물인식이 필수적이다. 따라서 차선인식에 라즈베리파이와 아두이노 나노를 사용하고,
추가적으로 사물인식(신호등 색상포함)은 초음파 센서를 이용한다.

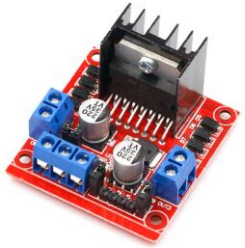


02 차량구동 - 설계

사용 부품

라즈베리파이 4 B

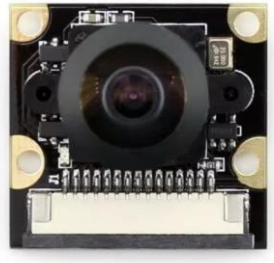
n20 모터



L298n모터드라이버

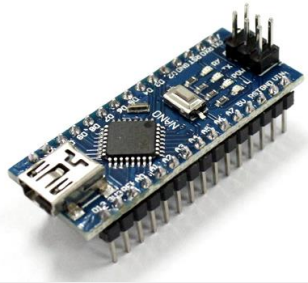
18650 배터리 & UPS

웹카메라



광각카메라

초음파센서



아두이노 나노

MG90S 서보모터

스탭업 모듈

조도센서

12V LED

보조배터리

라즈베라파이 & 아두이노 & 사물인식과 &차선인식을 고려하여 위의 부품들을 사용해서 설계

Auto
nomous

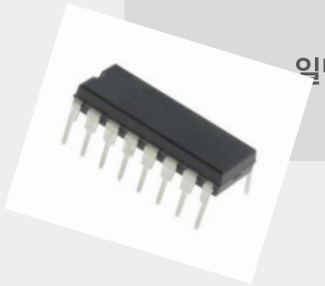
02 차량구동 - 설계

아두이노 나노 보드 사용 이유

1. 서보모터

서보모터를 라즈베리파이 GPIO로 제어할 때 지터링 현상이 발생할 수 있다. 이를 해결하기 위해 pigpiod 라이브러리를 사용할 수 있지만, 고장이 쉽게 날 수 있다는 단점이 있다.
따라서 하드웨어적으로 지터링 현상을 해결하기 위해 아두이노 보드를 활용하기로 했고, 그중 소형 보드인 아두이노 나노를 사용하기로 결정했다.

2. 조도센서



일반적인 조도센서를 라즈베라파이에서 작동 시키기 위해서는 아날로그 신호를 디지털 신호로 변환시켜주는 MCP3008과 같은 부품이 필요하다.
하지만 아두이노를 사용한다면 디지털 신호를 받는 핀이 있기때문에 위와 같은 부품이 필요없다.

3. 방열

방열을 위해 라즈베리파이를 상단에 위치시키기 위해서는 아두이노를 사용해 모터와 LED를 제어하고 하는 방법이 제일 적합하다고 판단했다.

02 차량구동 - 배터리

Ups + 보조배터리

라즈베리파이 4B

5V 3A (권장)
5V 1.5A (실제)

n20 모터 2개

6V 1A (권장)
9V 0.7A (실제)

서보모터

5V 1A (권장)
5V 1A (실제)

12V LED

12V 0.7A (권장)
9V 0.7A (실제)

5V 4A UPS

라즈베리파이 + 기타 부품

5V 3A UPS

기타부품

+

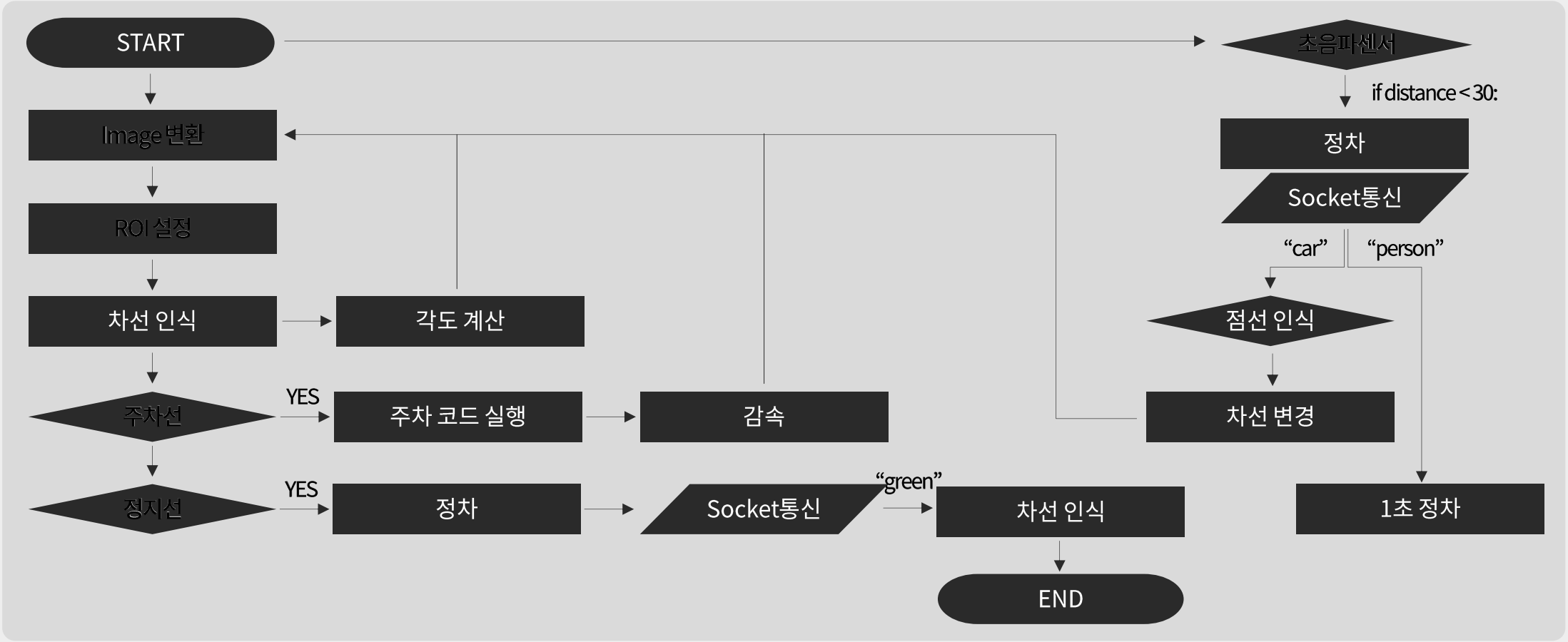
20W 보조배터리

라즈베리파이

Auto
nomous

03 차선인식

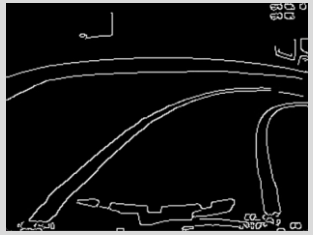
라즈베리파이에서 작동하는 코드의 전체 흐름도
정지선을 인식할 때(신호등)와 초음파 센서로 거리를 인식했을 때(차/사람) socket통신을 진행함으로써 불필요한 통신으로 인한 속도저하 현상을 방지했다.



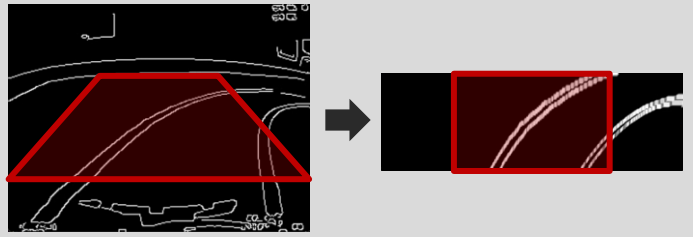
03 차선인식

OpenCV를 활용한 Canny Edge 검출 방식과 sliding window 추적 기술을 활용

Original Frame



Canny



Warp

Bird-Eye View

차선용 ROI 설정

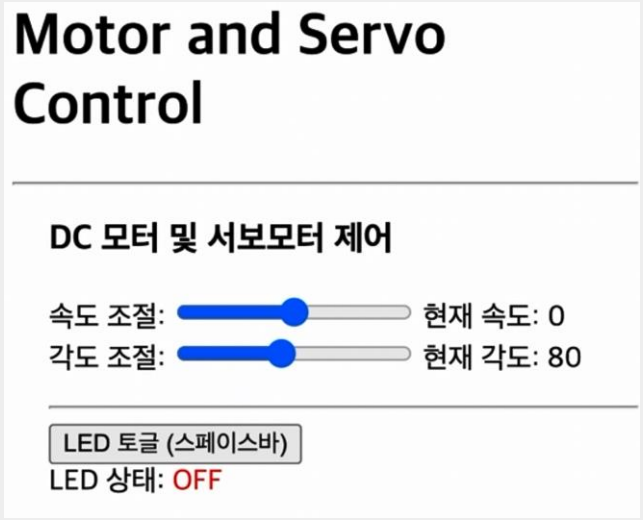
Sliding Window

차선인식을 통해 얻은 차선의 중앙값과 차량의 중앙값을 비교하여 deviation값을 계산

Auto
nomous

03 차선인식

Flask를 활용하여 수동조작을 진행

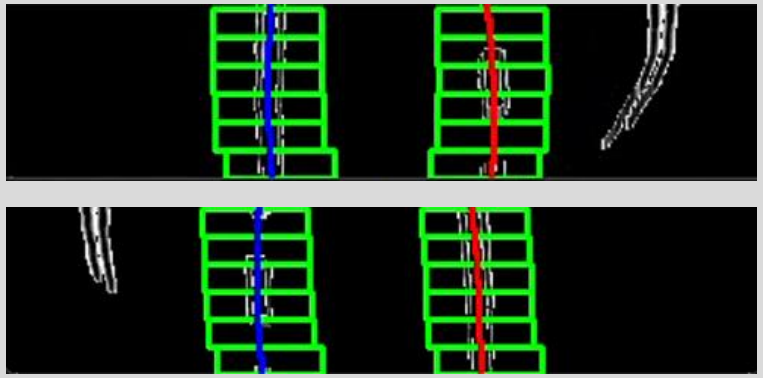



```
93
94   k_values = [150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40]
95   j_values = [-170, -150, -120, -100, -80, -20, 5, 30, 60, 120, 140, 160]
96
```

수동조작을 이용하여 측정한 각도를 얻고,
선형보간법을 사용하여 deviation값과 서보모터의 각도의 관계 함수를 도출함으로써 서보모터를 1도 단위의 각도로 움직일 수 있도록 했다.

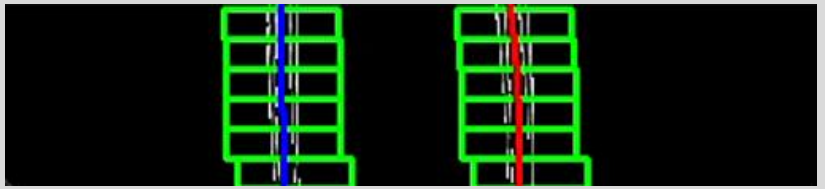
03 차선인식

Line() 클래스의 is_dashed 속성을 추가하여 검출한 차선이 실선인지 혹은 점선인지 판단했다. 이를 통해 차량 장애물의 개수와 위치에 상관없이 차선변경이 가능하게 구현했다.



점선

위: 오른쪽 점선 (비어있는 window 2개)
아래: 왼쪽 점선 (비어있는 window 3개)



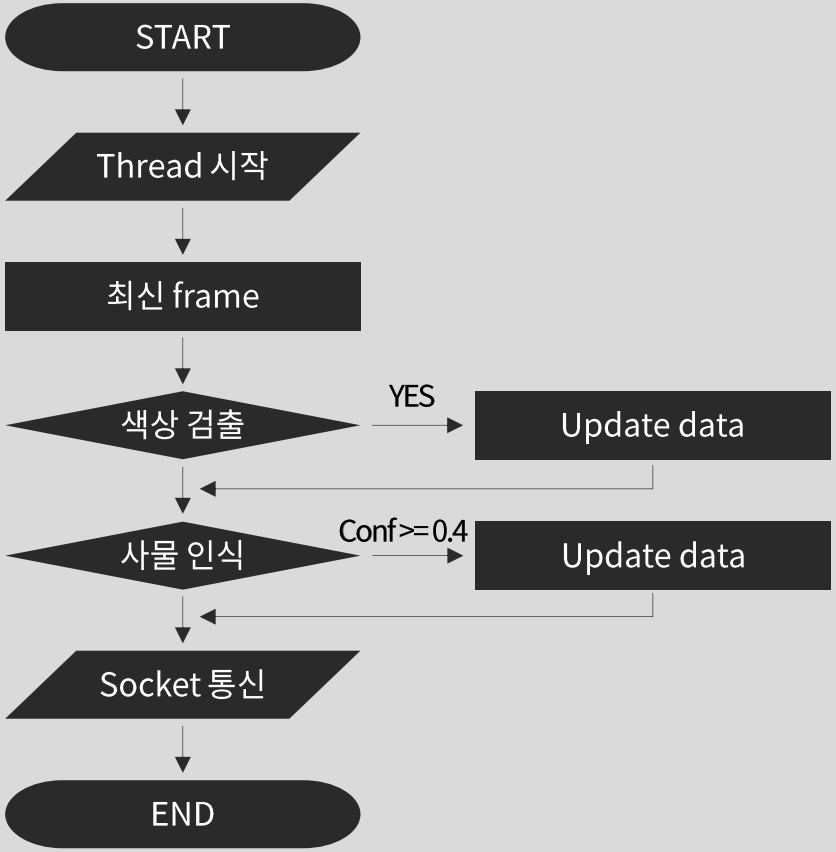
실선

비어있는 window 0개

minpix보다 적은 수의 pixel이 검출되는 window의 수를 count,
총 6개 중 2개 이상의 window가 비어 있는 경우를 점선으로 간주한다.

03 사물인식

PC에서 작동하는 코드의 전체 흐름도

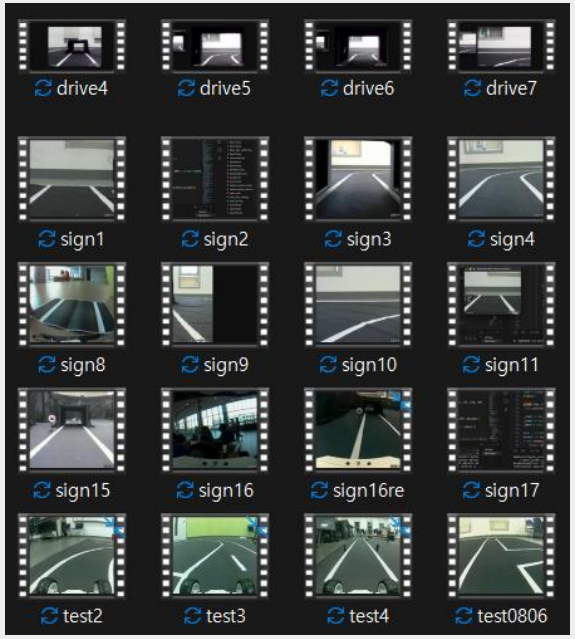


색상 검출을 진행한 후에 사물 인식 결과값을 덮어씌움으로써 사물인식으로 신호등을 검출 못할 경우를 대비하였다. 이는 초기 신호등에서 초록불의 조도가 약해 사물인식의 정확도가 낮았기 때문이다. 사물인식의 정확도가 0.4인 값만 유효하게 설정함으로써 정확도를 높였다.
thread를 이용해서 비디오 스트리밍과 객체 검출이 서로 방해되지 않고 동시에 수행할 수 있게 하여 딜레이를 최소화했다. video_stream() 함수가 독립된 쓰레드에서 실행되기 때문에, 메인 루프는 비디오 스트리밍의 지연 없이 계속해서 객체 검출과 처리 작업을 수행할 수 있다.

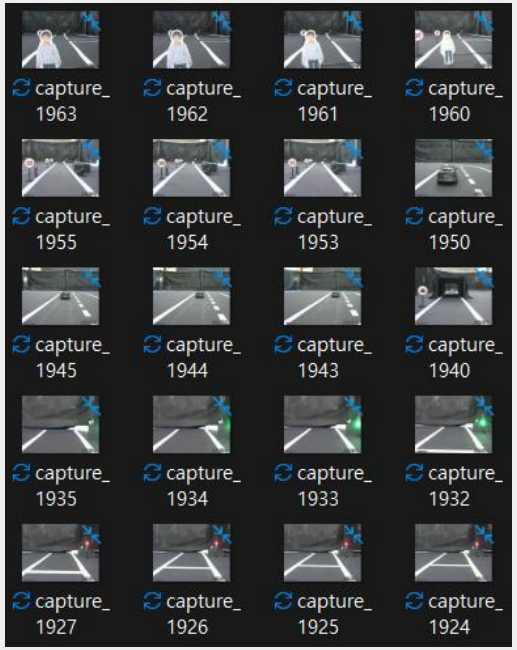
```
23  def video_stream():
24      global frame
25      cap = cv2.VideoCapture('http://192.168.2.2:8081/')  # 웹캠 URL을 설정
26      while True:
27          ret, latest_frame = cap.read()
28          if not ret:
29              break
30          frame = latest_frame  # 최신 프레임 저장
31      cap.release()
32
33  > def find_green(img, size = 10): ···
45
46  > def find_red(img, size = 10): ···
62
63  # 비디오 스트리밍을 위한 쓰레드 시작
64  thread = threading.Thread(target=video_stream)
65  thread.start()
```

03 사물인식

연습 주행 영상을 통한 학습 데이터 수집



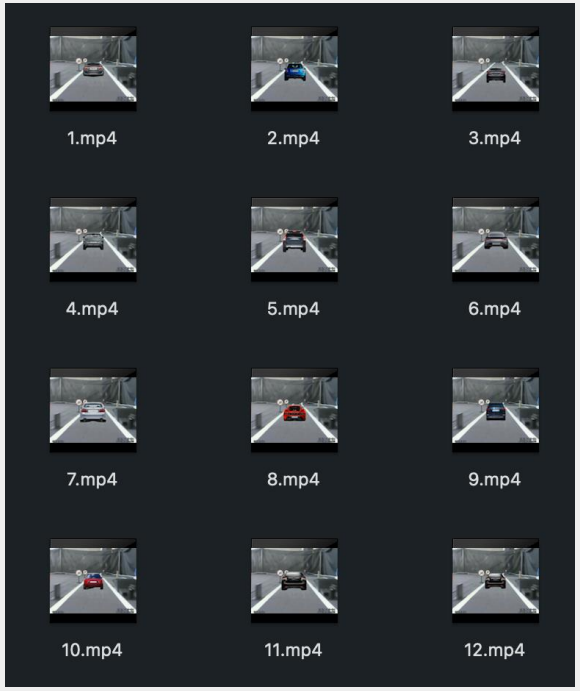
```
video_path = 'ref/12.mp4'
save_directory = 'dataset'

# 저장할 디렉토리가 없으면 생성
if not os.path.exists(save_directory):
    os.makedirs(save_directory)

# 비디오 캡처 객체 생성
cap = cv2.VideoCapture(video_path)
frame_count = 2108

while cap.isOpened():
    ret, frame = cap.read()
    frame = cv2.resize(frame, (320, 240), interpolation=cv2.INTER_AREA)

    if not ret:
        print("비디오를 불러오는데 실패했거나 비디오가 끝났습니다.")
        break

    cv2.imshow('Video', frame)

    # 키 입력 대기 (1ms)
    key = cv2.waitKey(1) & 0xFF

    # 'c' 키를 누르면 현재 프레임을 캡처하여 저장
    if key == ord('c'):
        frame_count += 1
        filename = f"{save_directory}/capture_{frame_count}.jpg"
        cv2.imwrite(filename, frame)
        print(f"캡처 저장: {filename}")

    # 'q' 키를 누르면 종료
    elif key == ord('q'):
        break
```



영상을 캡쳐하는 코드를 별도로 작성해서 주행 영상을 frame 단위로 custom model을 위한 학습데이터화 했다.

03 사물인식

영상편집을 활용한 차량 데이터 수집

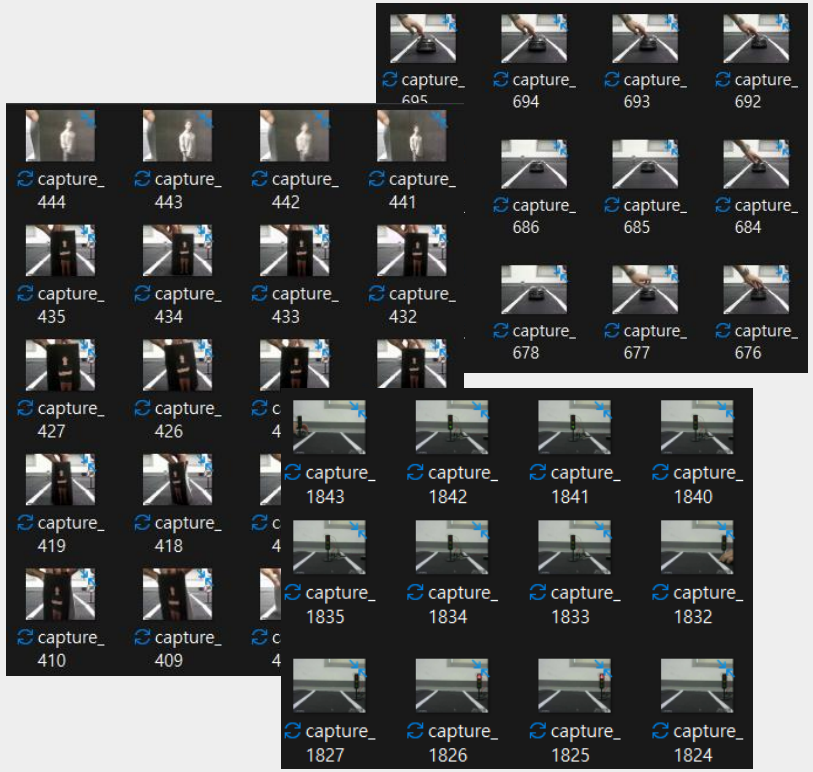


실제 환경에서 다양한 각도로 많은 차량 데이터를 추가적으로 얻을 수 있었다.

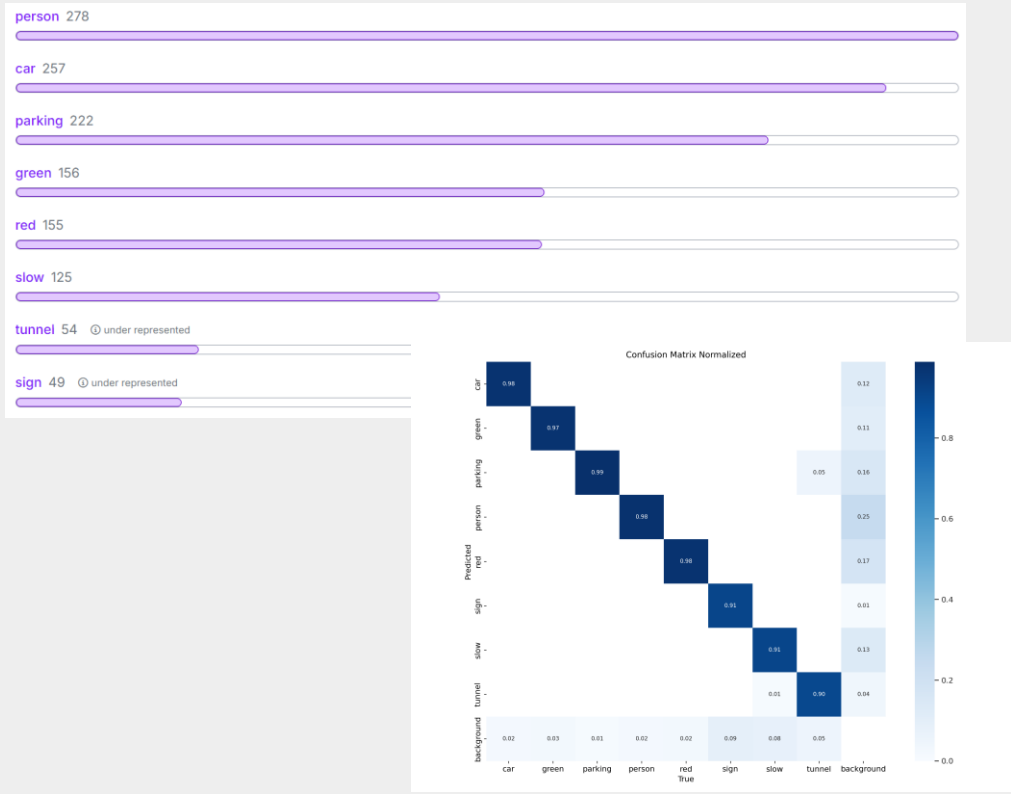
그 결과 차량을 잘 인식하여 차선변경을 잘 진행 시킬 수 있었다.

03 사물인식

주요 클래스들의 학습 데이터 추가 수집



사람 이미지 출력하여 각도별로 촬영, 차량모형을 각도별로 촬영, 신호등 위치별로 촬영하여
다양한 각도로 많은 데이터를 추가적으로 얻을 수 있었다.

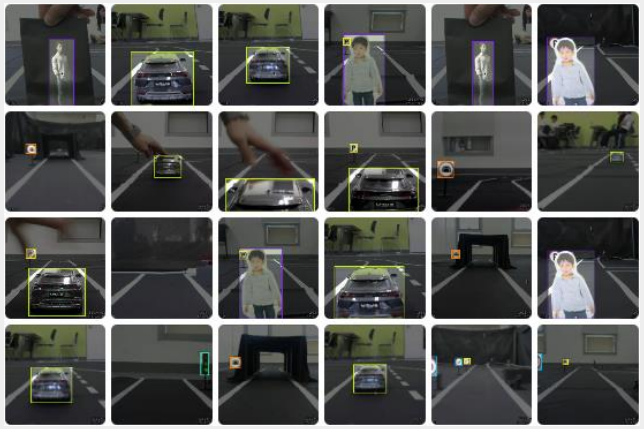


그 결과 주요 클래스인 person, car, green의 학습데이터를 충분히 확보할 수 있었고,
학습 결과 높은 정밀도와 재현율을 보였다.

03 사물인식

Roboflow에서 라벨링과 이미지 전처리 과정을 진행한 후, Google colab에서 YOLOv10 custom model 학습을 진행
차량이 진행할 때 진행방향에 따라 장애물의 각도가 달리 보일 수 있기 때문에 horizontal shear를 추가하여 학습데이터의 다양성을 높였다.

Preprocessing	Auto-Orient: Applied Resize: Stretch to 640x480
Augmentations	Outputs per training example: 3 Shear: ±15° Horizontal, ±0° Vertical



```
results = model.train(data="/content/data.yaml", epochs=100, batch=64, patience=20, save_period=25, resume=True,
                      fliplr=0.0, project=drive_dir, iou=0.5, translate=0.4, name='yolov10n_training')
```

Colab 링크 : https://colab.research.google.com/drive/1HrctzptCwxvjrN_cVvCVcRGWvTKBJpng#scrollTo=y-lhbe5CPwYV

Auto
nomous

03 사물인식

YOLOv10 custom model 학습에 사용한 하이퍼파라미터
박스 위치는 중요하지 않다는 점과 빠르게 움직이고 부분적으로 보이는 물체를 감지하는 것에 초점을 맞춰 값을 조정

Parameter	Default	Value	이유
epochs	100	100	100회 이상인 경우 성능이 미세하게 높았으나, 100회만으로 충분히 원하는 결과를 얻을 수 있었음
batch	16	64	Batch size가 낮을수록 인식률이 안 좋았음. 실험으로 얻은 최상의 결과값임
patience	100	20	epochs 숫자에 맞춰 값을 낮춰 설정함으로써 과적합을 방지함
save_period	-1	25	모델 체크포인트 저장 빈도를 설정하여 런타임 에러로 인한 학습중단에 대비함
resume	False	True	마지막으로 저장한 체크포인트부터 훈련을 재개하여 런타임 에러로 인한 학습중단에 대비함
fliplr	0.5	0.0	좌우반전된 이미지가 불필요하기 때문에 해당 기능을 제외함
iou	0.7	0.5	값이 낮을수록 중복되는 상자를 제거하여 탐지 횟수가 줄어들어 중복을 줄이는 데 유용함
translate	0.1	0.4	부분적으로 보이는 물체를 감지하는 학습을 도와 근거리에 있는 사물 인식의 정확성을 높임

03 사물인식

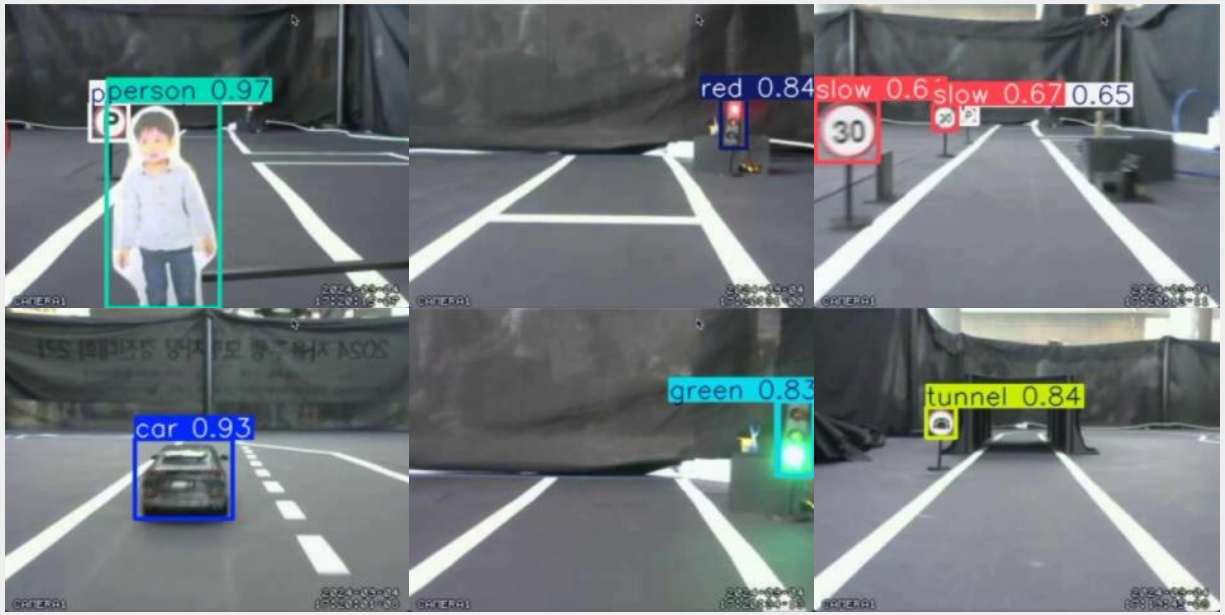
YOLOv10 custom model 학습 로그

No.	dataset	hyperparameters	mAP50
1	주행영상+인터넷 이미지	Epochs=50, batch=16	0.894
2	주행영상+인터넷 이미지	Epochs=50, batch=64	0.762
3	주행영상+자체제작 장애물	Epochs=100, batch=32	0.935
4	주행영상+자체제작 장애물	Epochs=100, batch=64, iou=0.5, translate=0.4	0.967
5	주행영상+자체제작 장애물	Epochs=200, batch=64, iou=0.5, translate=0.4	0.984
6	주행영상+자체제작 장애물	Epoch=100, batch=64, iou=0.5, translate=0.4	0.970
7	주행영상+자체제작 장애물	Epoch=100, batch=64, iou=0.5, translate=0.4, fliplr=0.0	0.979
8	주행영상+주최측 장애물	Epoch=100, batch=64, iou=0.5, translate=0.4, fliplr=0.0	0.984

03 사물인식

YOLOv10 custom 학습모델을 활용한 사물인식 결과

```
while True:
    # 버퍼를 클리어하여 최신 프레임만 읽음
    for _ in range(15):  # 15 프레임 정도를
        ret, frame = cap.read()
        if not ret:
            break
    frame = cv2.resize(frame, (320, 240),

    # YOLOv8 모델로 프레임 예측
    results = model(frame)

    # 결과를 이미지에 그리기
```

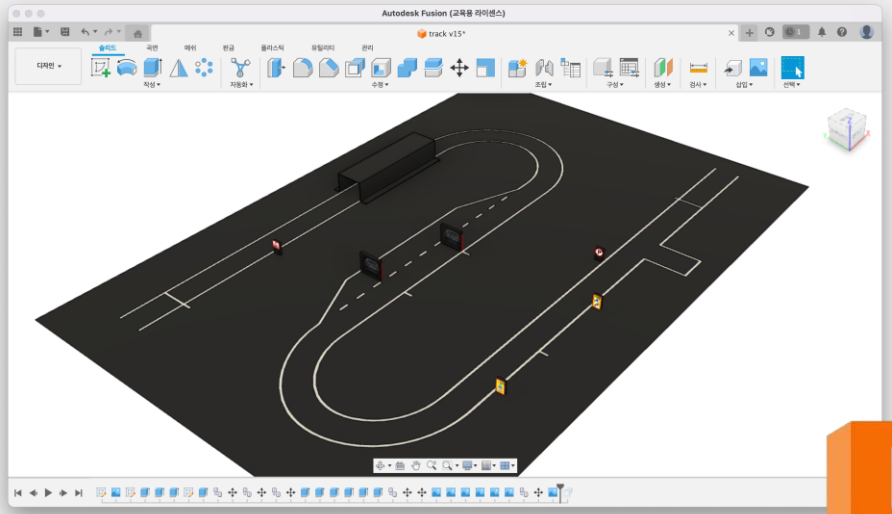


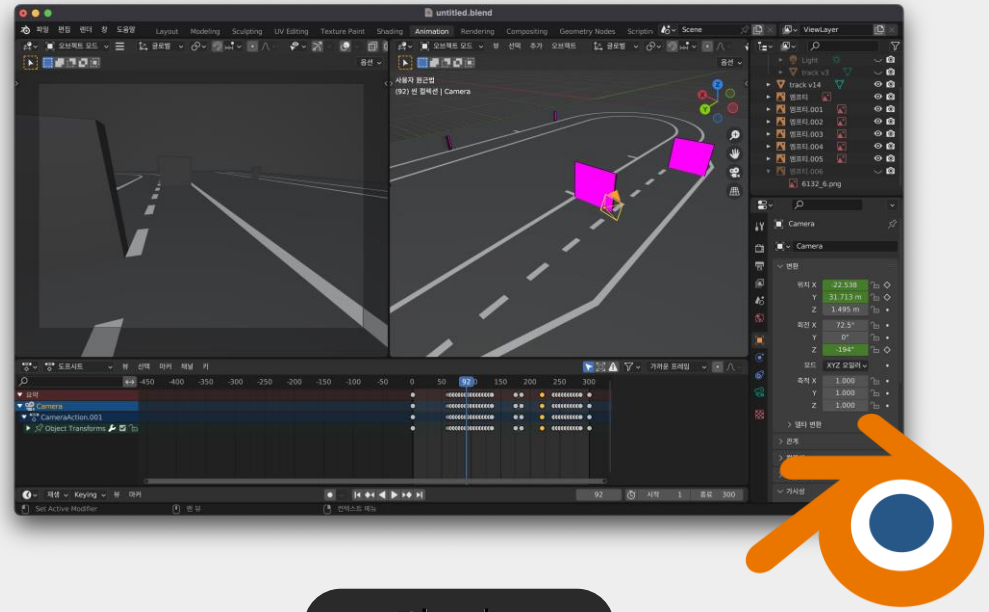
영상 : https://drive.google.com/file/d/1BiIczDAHSzV0vkQL8g0XUl47ae7JcTJz/view?usp=sharing

04 장애물 타파 방향성(연습용 트랙 설치 전)

시뮬레이션

Fusion360과 Blender를 활용하여 실제 주행과 비슷한 조건으로 시뮬레이션 제작





Fusion 360

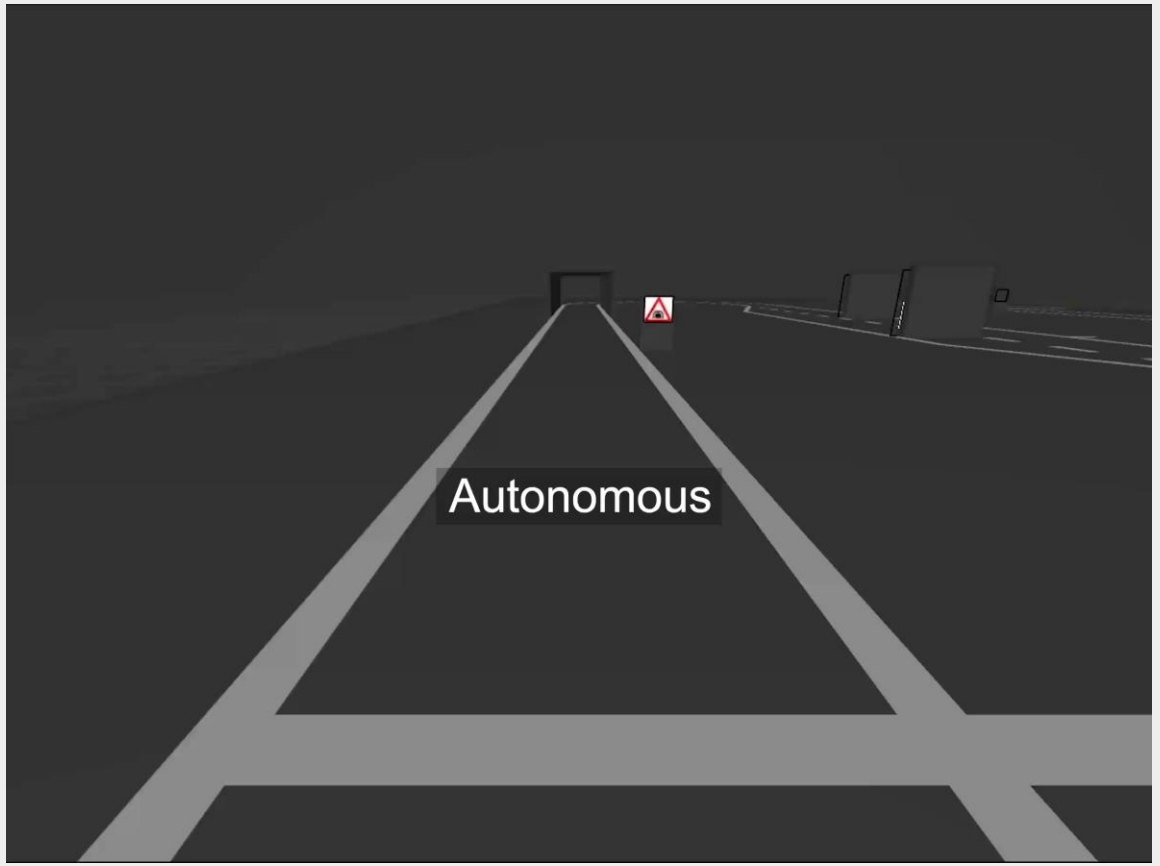
경진대회의 주행트랙과 장애물들을
3D모델링

Blender

카메라의 실제 높이와 각도, 광각 등을
고려하여 시뮬레이션 영상 제작

04 장애물 타파 방향성(연습용 트랙 설치 전)

시뮬레이션

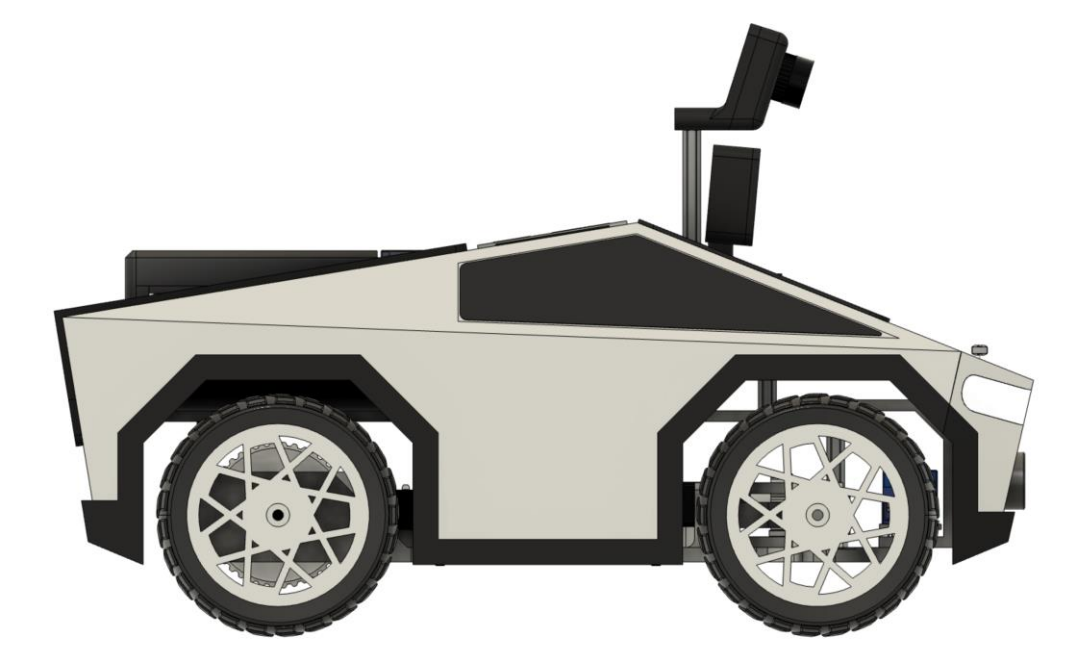


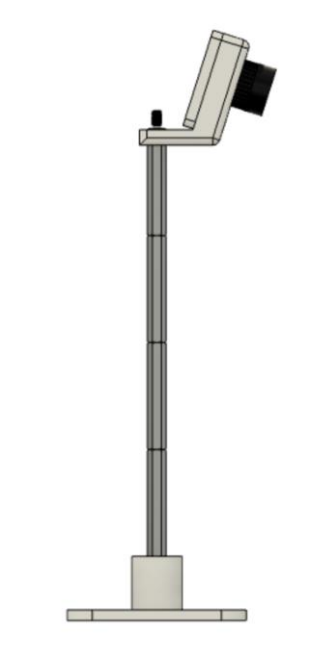
연습용 트랙 설치 전 위와 같이 실제 제작한 영상으로 장애물 타파 방향성을 구현

영상 : https://drive.google.com/file/d/1QKRVvUfMPGsWvJDHYlrAPJjik7w9mvPv/view?usp=sharing

04 장애물 타파 방향성(차량 제작 전)

실제테스트

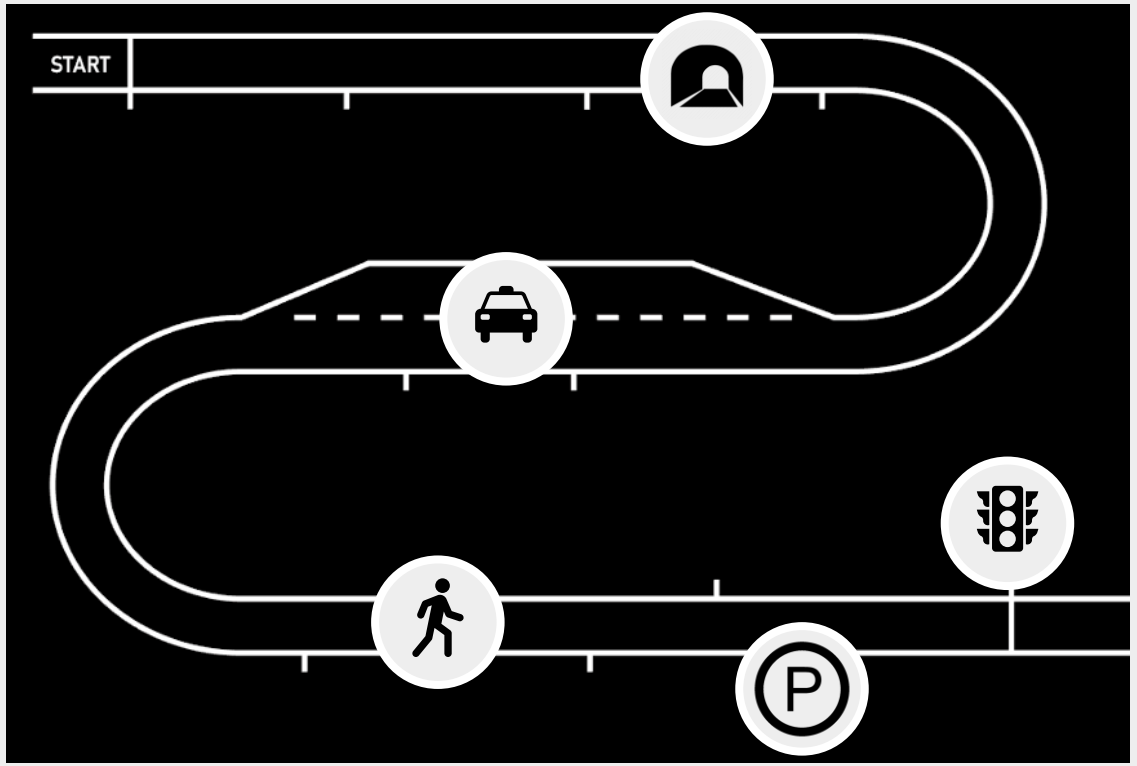




정확성을 높이기 위해 모델링한 차량을 바탕으로 스탠드형 카메라를
제작하여 실제 테스트를 진행

04 장애물 타파 방향성

장애물 타파 아이디어



터널

조도센서를 이용하여 어두워지면 LED 점등

정적장애물(차량)

사물인식(YOLO)과 초음파센서를 이용하여
동일 차선 내의 장애물을 인식하면 차선 변경

동적장애물(사람)

사물인식(YOLO)과 초음파센서를 이용하여
근거리의 사람이 인식되었을 때 긴급 정지

주차

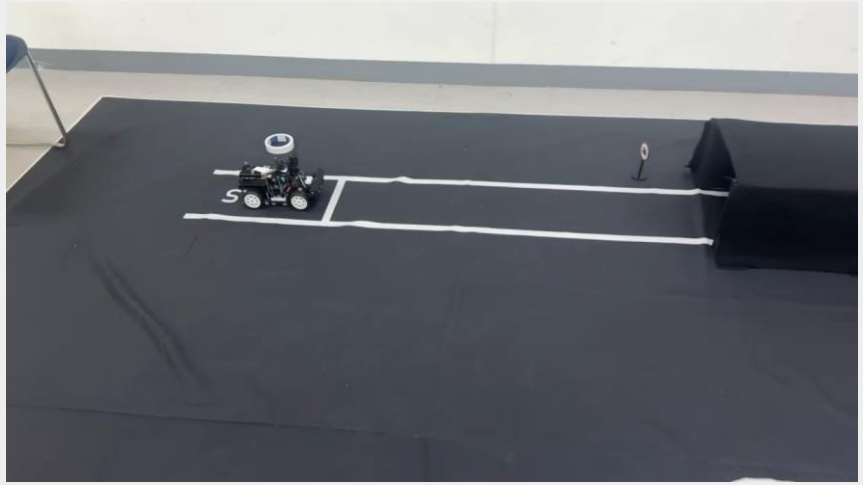
주차선(수평 평행선)을 인식하면 주차 코드
실행

신호등

정지선 검출 후 정차, 사물인식(YOLO)과
OpenCV 색상 검출을 통해 초록불이면 진행

04 장애물 타파 방향성 - 터널

조도센서를 이용하여 어두워지면 LED 점등



전면부 모습

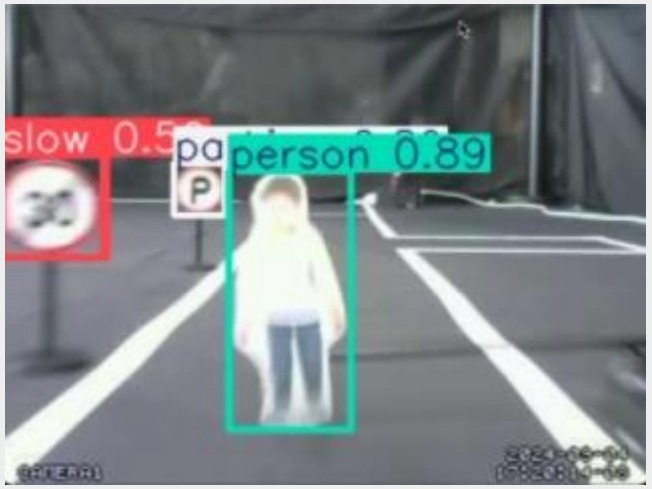
후면부 모습

영상(좌) : https://drive.google.com/file/d/1EwoJAxI3LC1ocNfb7VptGZVOvxzMP5Z5/view?usp=sharing
영상(우) : https://drive.google.com/file/d/1lz4IJESvCW45EQDqzXX1Ck8eEUjJinZi/view?usp=sharing

04 장애물 타파 방향성 – 차&사람

사물인식(YOLO)과 초음파센서를 이용하여 '차'일 경우, 동일 차선 내의 장애물을 인식하면 차선 변경 & 근거리의 사람이 인식되었을 때 긴급 정지

```python
try:
    distance1 = measure_distance(14, 15) #전방 센서
    # print(distance1)
    if distance1 > 30:
        set_servo_angle(angle)
        set_motors_speed(speed)
    elif 5 < distance1 < 30:
        set_motors_speed(0)
        try:
            data = client_socket.recv(1024).decode()
            if data:
                print(f"Received command: {data}")
                if data == "car":
                    if dir == 2:
                        print("go_right")
                        set_servo_angle(160)
                        set_motors_speed(speed)
                        time.sleep(0.3)
                    elif dir == 1:
                        print("go_left")
                        set_servo_angle(15)
                        set_motors_speed(speed)
                        time.sleep(0.3)
                elif data == "person":
                    set_motors_speed(0)
                    time.sleep(1)
                    speed = 20
        except socket.error as e:
            print(f"Socket error: {e}")
            break
    else:
        set_motors_speed(0)
        time.sleep(1)
```



동적장애물의 경우 초음파 센서로 어느 정도 거리를 인식하고 사물 인식 결과값을 받아온다.
그것이 '차'일 경우, 왼쪽과 오른쪽 중 어느 곳이 점선인지 파악한 다음 차선변경을 하는 코드를 작성했다.
'사람'일 경우, 사람 검출 시 멈추는 코드 주차와 이후 주차선 검출의 정확도를 높이기 위해 속도를 낮추는 코드를 실행시킨다.

04 장애물 타파 방향성 – 주차

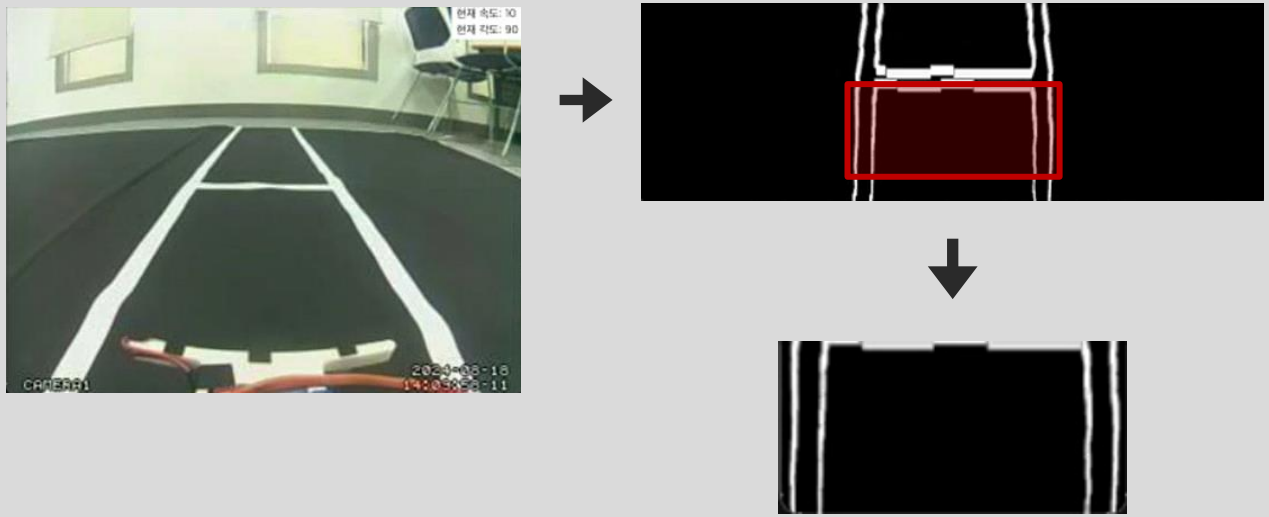
주차선(수평 평행선)을 인식하면 주차 코드 실행


```python
def detect_horizon(img):  # 주차선 감지
    global Parking
    if Parking:
        return Parking

    # cv2.imshow('warped frame for parking', img)
    lines = cv2.HoughLinesP(img, 1, np.pi / 180, threshold=100, minLineLength=50, maxLineGap=10)

    if lines is not None:
        count = 0
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = np.degrees(np.arctan2(y2 - y1, x2 - x1)) % 180

            if 170 <= angle or angle <= 10:
                count += 1

        if count >= 1:
            print("parkingline")
            parking_code()
            Parking = True

    return Parking
```

차선의 오른쪽 구간을 주차선 검출용 ROI로 설정하고,
HoughlinesP()를 이용하여 수평선을 감지하면 주차코드를 실행한다.

04 장애물 타파 방향성 – 신호등

정지선을 인식하면 잠시 멈춤

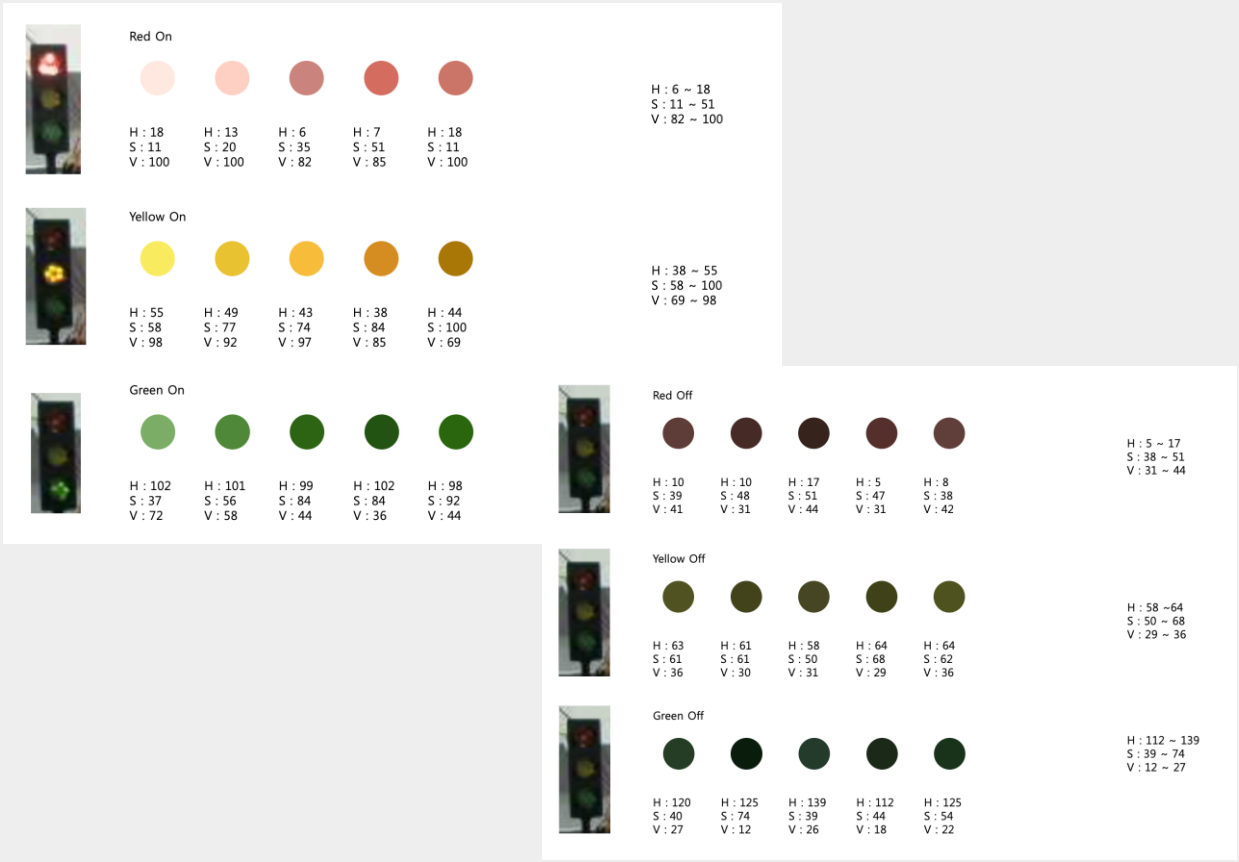



```python
def detect_stopline(img):  # 정지선 감지
    global Stop
    if Stop:
        return Stop

    # cv2.imshow('warped frame for stopline', img)
    lines = cv2.HoughLinesP(img, 1, np.pi / 180, threshold=100, minLineLength=30, maxLineGap=10)

    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = np.arctan2(y2 - y1, x2 - x1) * 180 / np.pi
            angle = angle if angle >= 0 else angle + 180
            if 170 <= angle or angle <= 10:
                set_motors_speed(0)
                print("stopline")
                time.sleep(1)
                Stop = True

    return Stop
```

차선의 중앙부분을 정지선 검출용 ROI로 설정하고,
HoughlinesP()를 이용하여 수평선을 감지하면 일시 정지한다.

04 장애물 타파 방향성 - 신호등

정지선 검출 후 정차, 사물인식(YOLO)과 OpenCV 색상 검출을 통해 초록불이면 진행



신호등 작동 영상에서 사진을 캡쳐, 색상을 추출하여 그 범위에 있는 값을 인식하도록 설정

영상 : https://drive.google.com/file/d/1rpoOY6faw0vga1dLsdxGICITDmCgGc0D/view?usp=sharing

04 장애물 타파 방향성 - 신호등

신호등 추출 방법 - 색상인식

	Red				Yellow				Green			
	On		Off		On		Off		On		Off	
	low	high	low	high	low	high	low	high	low	high	low	high
H(360)	2	20	5	17	38	55	58	64	142	182	112	139
S(100)	19	52	38	51	58	100	50	68	5	47	39	74
V(100)	84	100	31	44	69	98	29	36	94	100	12	27
H(360)	112	360										
S(100)	0	6										
V(100)	90	100										
H(180)	1	10	2.5	8.5	19	27.5	29	32	71	91	56	69.5
S(255)	48.45	132.6	96.9	130.05	147.9	255	127.5	173.4	12.75	119.85	99.45	188.7
V(255)	214.2	255	79.05	112.2	175.95	249.9	73.95	91.8	239.7	255	30.6	68.85
H(180)	56	180										
S(255)	0	15.3										
V(255)	229.5	255										

```python
def find_green(img, size = 10):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    green_lower = np.array([70, 10, 230], np.uint8)
    green_upper = np.array([95, 120, 255], np.uint8)
    green_mask = cv2.inRange(hsv, green_lower, green_upper)

    contours, _ = cv2.findContours(green_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        if cv2.contourArea(contour) > size * size:
            return "green"
    return None

def find_red(img, size = 10):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    red_lower1 = np.array([0, 25, 150], np.uint8)
    red_upper1 = np.array([10, 135, 255], np.uint8)
    red_lower2 = np.array([170, 130, 150], np.uint8)
    red_upper2 = np.array([180, 255, 255], np.uint8)
    red_mask1 = cv2.inRange(hsv, red_lower1, red_upper1)
    red_mask2 = cv2.inRange(hsv, red_lower2, red_upper2)
    red_mask = cv2.bitwise_or(red_mask1, red_mask2)

    contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        if cv2.contourArea(contour) > size * size:
            return "red"
    return None
```

추출해 얻은 값을 변환하고 그 값을 바탕으로 코드를 작성
contourArea(contour)를 계산하는 과정을 추가하여 불필요한 검출을 예방했다.

04 장애물 타파 방향성 - 신호등

신호등 추출 방법 - 사물인식

```
while True:
    # 버퍼를 클리어하여 최신 프레임만 읽음
    for _ in range(15):  # 15 프레임 정도를
        ret, frame = cap.read()
        if not ret:
            break
    frame = cv2.resize(frame, (320, 240),

    # YOLOv8 모델로 프레임 예측
    results = model(frame)
```

YOLO custom 학습모델을 통해 초록불이 들어올 경우, “green”을 반환하고 일정 시간동안 직진

영상 : https://drive.google.com/file/d/1PJYiHaX7-K4VM-55ozGs7wRrHfJqW-qK/view?usp=sharing

Autonomous 설계발표

THANK YOU

백재민 | 박준아

Auto
nomous