

제5회 국민대 자율주행 경진대회 예선경기 #과제3

자율주행 SW 설계서

1. 참가팀 일반사항

참가팀	팀명	심마니
	팀장	임인섭(건국대)
	팀원	공경은(건국대), 김주빈(건국대), 오민혁(건국대), 조준하(건국대)

2. 미션 완수를 위한 자율주행 SW 설계서

111111

수행 미션 (1번)	차선을 벗어나지 않고 주행 <ul style="list-style-type: none"> 주행트랙에서 차량이 차선을 벗어나지 않고 주행해야 함. 코너와 곡선구간 차선 위에 세워둔 차선이탈 판정용 블록을 쓰러뜨리지 않아야 함. 	
① 사용하는 센서별 용도	카메라	영상처리를 통해 차도에서 양쪽 차선의 위치를 파악하기 위해 사용
	초음파센서	주행 중에 전방에 있는 장애물과 충돌하는 것을 막기 위해 사용
	라이다	SLAM 기술로 차량이 트랙의 어디쯤 왔는지 위치를 파악하기 위해 사용
	IMU	차량의 현재 속도를 측정하기 위해 사용
② 구현할 기능 요약, 그리고 구체적인 구현방안	(1)	트랙에 그려진 차선을 인식하고 차량이 차선의 중간쯤에서 달려야 함.
	<ul style="list-style-type: none"> - 카메라 ROS 토픽을 수신하여 OpenCV 이미지로 변환하여 영상처리를 진행한다. - 칼라 이미지를 회색톤으로 변환하고 다시 이진화 작업을 거쳐 차선은 흰색으로 나머지는 모두 검은색으로 표시하는 이미지를 만든다. - 화면의 아래부분을 관심영역(ROI)으로 설정하여 이 부분만 처리한다. - 흰색점이 많이 몰려 있는 구역을 찾고 그곳을 차선이 있는 곳으로 지정한다. - 왼쪽차선과 오른쪽차선을 각각 찾고, 해당 위치를 저장한다. X좌표값만 저장한다. - 화면의 중심을 기준으로 왼쪽차선과 오른쪽 차선이 동일한 거리만큼 떨어져 있으면 현재 차량이 차선의 중앙을 달리고 있는 것으로 파악한다. - 화면의 중심 기준선에서 왼쪽차선이 오른쪽 차선보다 멀리 있으면 핸들을 왼쪽으로 꺾어 왼쪽차선이 있는 쪽으로 차량의 방향을 튼다. 	

	<ul style="list-style-type: none"> - 화면의 중심 기준선에서 오른쪽차선이 왼쪽차선보다 멀리 있으면 핸들을 오른쪽으로 꺾어 오른쪽차선이 있는 쪽으로 차량의 방향을 튼다. - 트랙 중앙에 점선으로 그려진 차선은 제외하고 좌우에 실선으로 그려진 차선만 인식해야 한다. 점선을 왼쪽차선 또는 오른쪽차선으로 인식하는 경우를 막는다.
(2)	직선 구간에서는 빨리, 곡선 구간에서는 천천히 달려야 함.
	<ul style="list-style-type: none"> - 핸들의 조향각을 살펴서 좌우 꺾임 각도가 5도 이하이면 직선이라고 간주하고 속도를 크게 높인다. - 꺾임 각도가 5도 이상이면 곡선 구간이라고 간주하고 속도를 늦춘다. - S자 곡선처럼 중간에 회전방향의 좌우가 바뀌면서 직선구간이 잠깐 등장하는 경우가 있으므로 약간의 시간을 두고 여러 번 관찰하는 방식으로, 핸들의 꺾임 정도를 정확하게 파악하여 반응하도록 한다. - 갑작스러운 가속보다는 속도를 점진적으로 올리거나 내리는 방식으로 차량의 속도를 제어한다. - 모터 배터리의 잔량에 따라 모터의 회전속도가 하드웨어적으로 변화하는 경우가 있으므로 IMU센서를 사용하여 실제 차량의 이동속도가 어떤지 주기적으로 체크하여 속도제어에 반영한다.
(3)	핸들을 너무 자주, 그리고 급격하게 꺾으면 안됨. (안정적으로 주행하게끔)
	<ul style="list-style-type: none"> - 속도가 빠른 고속도로 주행에서는 핸들을 조금만 꺾어도 차량이 휘청이게 되며, 잘 못하면 차체가 중심을 잃어 사고가 날 수도 있다. - 차량의 속도가 빠를 경우에는 조금씩 핸들을 좌우로 계속 왔다갔다 꺾는 방식의 운전은 피한다. 특정 값 이하의 작은 핸들조작은 Skip 하고, 그 이상의 핸들조작만 반영하는 방식으로 핸들을 조작한다. - 차량의 속도가 빠른 경우에 급격한 핸들 꺾임은 차량을 휘청이게 하므로 피한다. 기존 핸들 각도와 새로운 핸들 각도의 차이가 큰지 체크하여 임계값 이상으로 크면 (또한 차량의 속도가 빠르면) 적당한 가중치 값을 곱하여 (예를 들면 0.5) 핸들조작 각도를 일부러 작게 만든다. - 이런 상황이 일정 횟수 이상 반복되면 차량의 속도를 빨리 줄인다. 차량의 속도가 줄면 가중치 값을 곱할 필요가 없으므로 핸들 조작량이 정상으로 되돌아 와서 차량이 차선을 벗어나지 않도록 신속한 핸들링이 가능해질 것이다.
(4)	차선 인식에 실패하면 원래 가던 방향으로 계속 주행해야 함.
	<ul style="list-style-type: none"> - 여러 가지 이유로 차선의 인식에 실패하면 일단은 차량이 원래 가던 방향으로 그대로 갈 수 있도록 앞서의 핸들 조향각과 속도를 그대로 사용한다. 이를 위해 핸들 조향각과 속도 값은 특정 변수에 저장하고 있어야 한다. - 하지만 정해진 시간이 지난 후에도 계속 차선인식을 하지 못하는 경우에는, 아마도 차선을 이탈했을 가능성이 높으므로, 이때에는 핸들을 정면으로 하고 속도를 0 값으로 하여 차량을 정차시킨다.
(5)	코너에 진입하기 전에 속도를 늦추고 코너에 진입한 후엔 속도를 높여야 함.
	<ul style="list-style-type: none"> - 고속 주행시에는 코너에 진입하기 전에 속도를 늦추는 것이 타당하다. 그리고 일단 코너에 진입하여 핸들이 꺾인 상태가 되면 속도를 다시 높여도 된다.

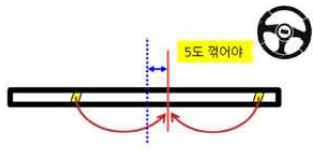
	<ul style="list-style-type: none"> - 카메라 영상에 윗부분을 살피서, 즉 조금 먼 곳에 있는 차선을 살피서 코너구간으로 진입하게 될 것인지를 파악하고, 이를 통해 코너에 진입하기 전에 차량의 속도를 늦췄다가 핸들을 꺾은 후 곧바로 다시 속도를 높인다. - 이 운전법은 일정 속도 이상의 고속에서만 의미가 있으므로 차량의 현재 속도를 파악하여 적절히 적용한다. <div style="background-color: yellow; padding: 5px;"> <p>(6) 장애물과 충돌하면 안됨.</p> </div> <ul style="list-style-type: none"> - 전방에 있는 장애물과의 충돌은 어떤 경우에도 피해야 하므로 초음파센서를 사용하여 전방 30센치 안에 장애물이 있는지 항상 체크한다. - 장애물 체크 시점은, 영상처리 작업을 통해 핸들 조향각과 차량의 속도를 결정한 후 모터제어 토픽을 발행하기 직전에 수행하면 된다. - 전방 장애물이 감지되는 경우에는 모터제어 토픽에서 속도 값을 0으로 세팅하는 방법으로 장애물 충돌을 방지함.
<p>③ 기타 구현 기술과 관련된 내용 자유롭게 기술</p> <p>(구현 기술과 관련된 그림 포함)</p>	<ul style="list-style-type: none"> • OpenCV 영상처리 기반 허프변환 기법을 기반 차선인식 (참조사이트 https://machinelearningknowledge.ai/lane-detection-tutorial-in-opencv-python-using-hough-transform/) <div style="display: flex; flex-wrap: wrap;">  </div> <ul style="list-style-type: none"> • OpenCV 영상처리 기반 원근변환과 슬라이딩윈도우 기법을 이용한 차선인식 (참조사이트 https://medium.com/geekculture/advanced-techniques-for-lane-finding-self-driving-cars-fc1a147fc49a) <div style="text-align: center;">  </div>

Image Read	카메라 영상신호를 이미지로 읽기
Warping	원근변환으로 이미지 변형
Gaussian Blur	노이즈 제거
Threshold	이진 이미지로 변환
Histogram	히스토그램에서 차선 위치 추출
Sliding Window	Sliding Window 좌우에 9개씩 쌓기
Polyfit	2차 함수 그래프로 차선 그리기
차선 영역 표시	차선영역 색칠하기 + 역변환
이미지 오버레이	원본 이미지에 차선영역 겹쳐 그리기



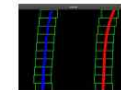
4점을 지정해 warping 실행



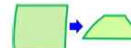
Warping 후



히스토그램



Sliding Window
2차 함수 그리기



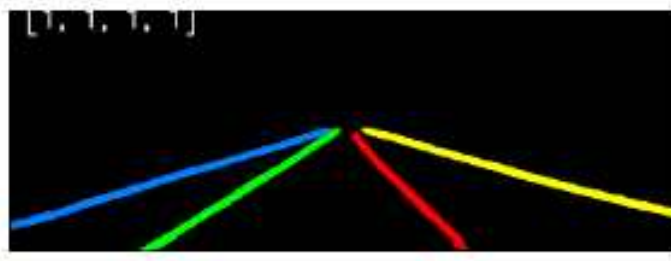
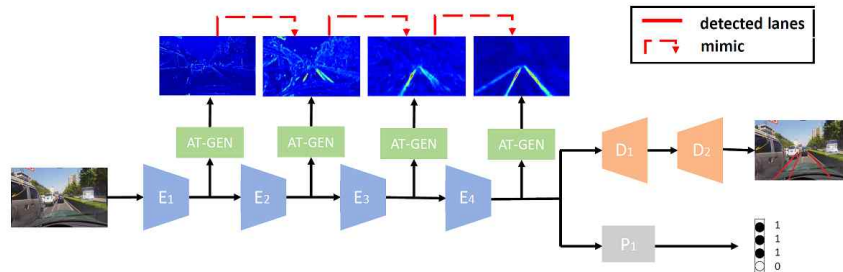
영역색칠/역변환



차선 영역 그리기

- 핸들 조향각 결정
- 핸들 꺾기~

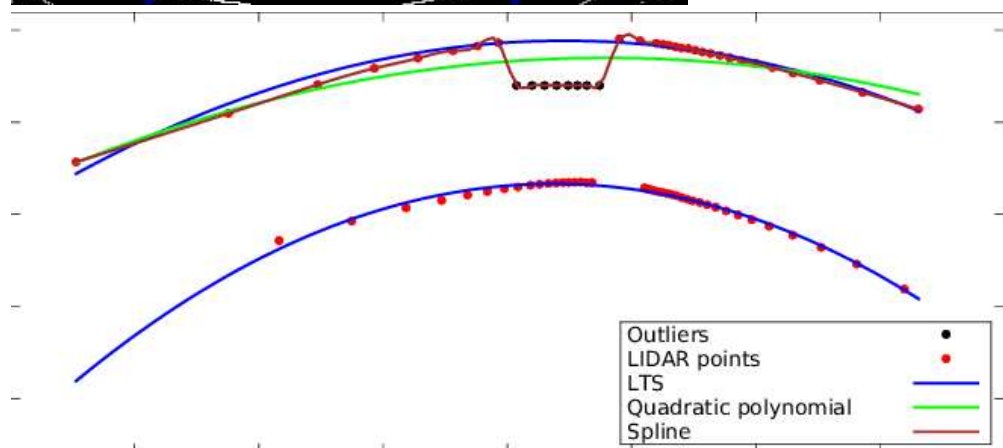
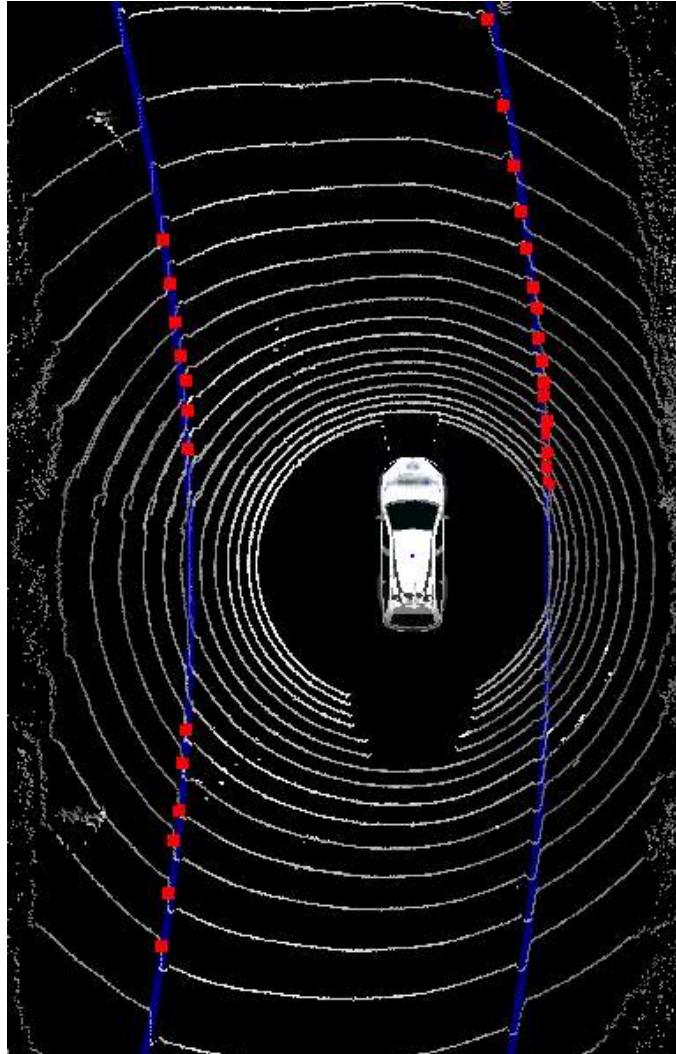
- 머신러닝 기반의 객체인식 모델을 이용한 차선인식
(참조사이트 https://github.com/InhwanBae/ENet-SAD_Pytorch)



수행 미션 (2번)	어두운 터널을 통과 <ul style="list-style-type: none"> 카메라 영상으로 차선을 인식하는 것이 불가능한 어두운 터널 안에서 거리센서(초음파센서 또는 라이다)를 이용해서 벽과 충돌하지 않으면서 주행하여 터널을 빠져나와야 함. 	
① 사용하는 센서별 용도	카메라	일반 주행 모드와 터널 주행 모드를 구분 하는데 사용한다.
	초음파센서	터널 벽면과 실제 거리를 계산하여 충돌 방지 및 경로 최적화
	라이다	차량 주위의 360도 방향의 터널 벽면을 탐지하여 경로 생성
	IMU	차량의 현재 상태를 반환
② 구현할 기 능 요약, 그리 고 구체적 구 현방안	(1)	카메라 없이 터널 내에서 경로를 제작해야 한다.
		<ul style="list-style-type: none"> 2d lidar에서 나오는 angle 및 distance 값을 point 값으로 변환하여 차량 위치를 원점으로 한 벽면의 좌표값을 반환한다. 반환된 좌표값들을 차량 위치를 기준으로 왼쪽, 오른쪽으로 나눈다. 나누어진 좌표값들 활용하여 좌,우측의 좌표값들을 각각 Pseudo Inverse Matrix를 활용하여 3차 함수로 근사화 한다 근사화된 3차 함수를 활용하여 중심 차선을 찾는다
	(2)	주어진 경로를 기반으로 angle 및 speed를 제어해야 한다.
		<ul style="list-style-type: none"> 주어진 경로 함수를 활용하여 Pure Pursuit method를 활용하여 제어를 진행한다. 터널 내에선 속도와 look ahead distance를 줄여 보다 안정적으로 주행하도록 한다.
	(3)	차량이 터널 벽면 한쪽에 치우쳐질 가능성이 있다.
		<ul style="list-style-type: none"> 초음파 센서를 활용하여 차량 좌우측 방 벽면과의 거리를 계산한다. 계산한 좌우측 거리 차이를 P제어를 활용하여 차량이 터널에 중심에 위치하도록 제어한다.
	(4)	장애물과 충돌 예방 노드 구현
		<ul style="list-style-type: none"> 측 후방 초음파 센서를 활용하여 일정거리 이하의 장애물이 있는지 항상 체크한다 장애물이 탐지되는 초음파 센서 반대 방향으로 조향각을 제어한다
	(5)	터널 주행 모드와 일반주행 모드를 구분 해야한다
		<ul style="list-style-type: none"> 카메라 이미지의 히스토그램을 활용하여 특정 임계값 이하의 pixel값이 많아지면 터널로 진입했다고 판단한다. 라이다의 point array는 항상 subscribe 하고 있다. 터널 주행모드로 바뀌면 위 (1)에서 (4)의 과정을 수행한다. 카메라의 이미지의 히스토그램에서 특정 임계값 이상의 pixel값이 많아지면 터널을 빠져나왔다고 판단한다.

③ 기타 구현
기술과 관련된
내용 자유롭게
기술

(구현 기술과
관련된 그림
포함)



수행 미션 (3번)	장애물을 피해서 주행 <ul style="list-style-type: none"> 도로 위에 놓인, 차선 안쪽에 놓인 장애물과 충돌하지 않고 피해서 주행해야 함. 차선을 잠시 벗어날 수도 있으나 장애물을 모두 피한 후에는 정상적으로 차선 안쪽으로 되돌아와서 주행해야 함. 	
① 사용하는 센서별 용도	카메라	영상처리를 통해 양쪽 차선의 위치를 파악하기 위해 사용 Yolo를 활용한 object detection을 통해 도로 위의 장애물 탐지
	초음파센서	회피 시에 장애물과의 안전거리 유지 확보를 위해 사용
	라이다	전방에 있는 장애물을 탐지하기 위해 사용
	IMU	차량의 현재 속도와 Yaw값을 측정하기 위해 사용
② 구현할 기 능 요약, 그리고 구체적 구 현방안	(1)	라이다를 사용하여 전방에 있는 장애물을 탐지한다.
		<ul style="list-style-type: none"> 라이다 Ros 토픽을 수신하여 sensor_msgs/PointCloud 메시지 형식을 갖는 포인트 클라우드로 변환한다. 포인트 클라우드에서 인접한 포인트들은 세그먼트라는 하위 집합으로 묶는다. 세그먼트들은 확인하며 서로 병합할 수 있는지를 확인하고 가능하면 병합한다. 병합된 세그먼트가 일정 크기 이상이면 장애물로 판단하고 원형 형태로 추출한다. 이때 겹치는 원형 장애물은 다시 병합한다.
	(2)	카메라를 사용하여 전방에 있는 장애물을 탐지한다.
		<ul style="list-style-type: none"> 카메라 Ros 토픽을 darknetRos 노드에서 수신하여 object detection을 진행한다. 임베디드 플랫폼의 특성상 (Nvidia TX2 고려하여) 경량화 모델인 YoloV3-tiny를 사용한다. 미리 학습된 YoloV3-tiny 모델에는 해당 장애물에 대한 Class 정보가 없으므로 해당 장애물에 대한 Custom 학습을 진행한다. darknetRos 노드에서 Custom 학습을 진행시킨 YoloV3-tiny 모델을 통해 장애물을 탐지한다.
	(3)	라이다, 카메라를 통해 얻은 장애물 정보를 취합하여 최종 장애물을 선정한다.
		<ul style="list-style-type: none"> 라이다와 카메라를 Calibration을 통해 서로의 좌표계를 맞춰준다. 각자 받은 객체 정보를 통해 최종 장애물 정보를 취합하여 Topic으로 Publish해준다. 이때 라이다 기반의 객체 정보에 Gain값을 크게 두어 최종 장애물 정보에 반영한다.
	(4)	장애물이 탐지되면 차량의 속도를 줄이고 현재 인지하고 있는 차선을 기준으로 변경하려는 차선 쪽에 가상 차선을 생성한다.
		<ul style="list-style-type: none"> 장애물이 감지되면 일정 거리 안에 들어올 시에 속도를 일정 수준까지 줄인다. Sliding window를 통해 생성한 현재 중심 차선을 기준으로 변경하려는 쪽에 차선

offset을 더해 가상의 중심 차선을 생성한다.

(5) 가상의 중심 차선과의 횡 방향거리를 기준으로 가상 차선까지 남은 횡 방향거리를 피드백하며 가상 차선을 추종하여 차선 변경을 통해 장애물을 회피한다.

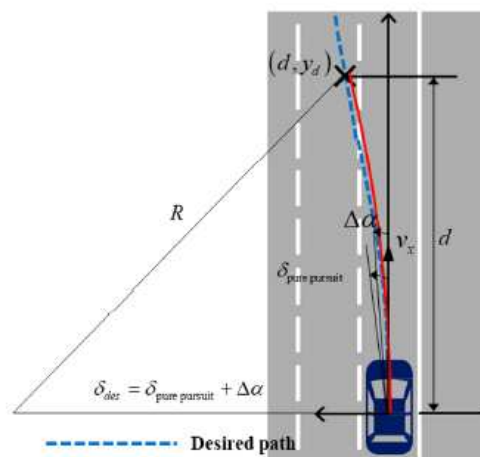
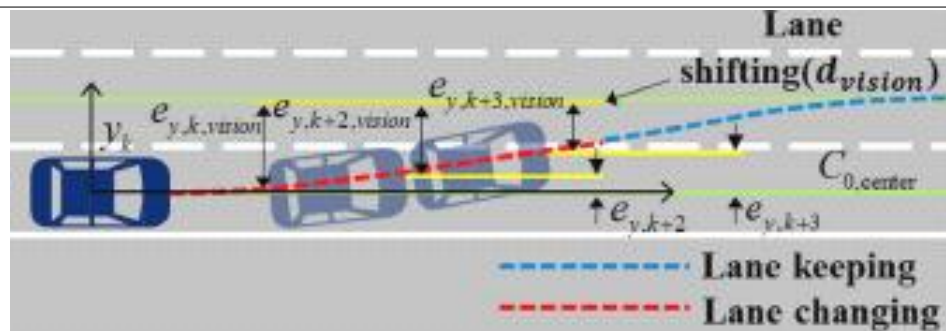
- 가상 중심 차선에서의 임의의 점 (x_1, y_1) 을 잡아 Pure Pursuit 알고리즘을 통해 진입 각 A 를 설정한다.
- 목표 진입각 A 를 Angle값에 반영한 후 현재 IMU 센서를 통해 받은 현재 속도 v 와, yaw값을 통해 횡방향속도 $v \cdot \cos(\pi/2 - \text{yaw})$ 를 구할 수 있다.
- 샘플링 시간을 dt 라 하면 횡방향거리 $v \cdot \cos(\pi/2 - \text{yaw}) \cdot dt$ 를 구할 수 있다.
- 가상 중심 차선과의 현재 차량의 거리를 e 라고 하면 $e = e - v \cdot \cos(\pi/2 - \text{yaw}) \cdot dt$ 를 통해 피드백하여 가상 중심 차선과의 거리를 줄이면서 차선을 변경하고 장애물을 회피한다.
- e 값에 비례하여 Angle값을 줄이고, e 값이 설정한 임계값 보다 작으면 변경한 차선에서의 Sliding window를 통해 차선을 생성하고 주행을 이어간다.

(6) (1)~(5)의 과정을 반복하여 장애물을 회피하며 주행을 이어간다. 차선 변경 중에 장애물이 자동차 측면과 너무 가까우면 Angle값에 margin을 준다.

- 차선 변경중에 측면에 있는 초음파 센서로 차량 옆면과의 거리를 구하고 너무 가까울시에 현재 Angle값에 margin을 준다.

③ 기타 구현 기술과 관련된 내용 자유롭게 기술

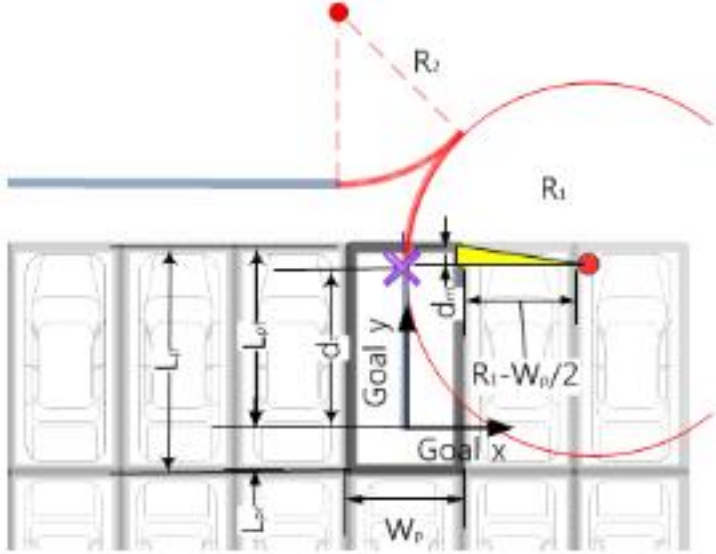
(구현 기술과 관련된 그림 포함)



수행 미션 (4번)	정지선 정차 후 일정시간 후에 다시 재출발하여 주행 <ul style="list-style-type: none"> 정지선을 발견하면 지나치지 않고 정지선 앞에 정차해야 함. 정차 후 일정시간(3초)이 지나면 다시 출발하여야 함. 한계시간(5초)이 지나도록 계속 정차해 있으면 안 됨. 	
① 사용하는 센서별 용도	카메라	영상처리를 통해 정지선을 검출하기 위해 사용
	초음파센서	
	라이다	
	IMU	현재 차량의 속도와 Yaw값을 파악하기 위해 사용
② 구현할 기 능 요약, 그리 고 구체적 구 현방안	(1)	정지선 검출을 위한 카메라 온 이미지를 전처리한다. <ul style="list-style-type: none"> 카메라에서 획득한 640*480사이즈의 이미지를 320*240사이즈로 리샘플링한다. 리샘플링을 통해 이미지의 크기를 줄임으로서 정지선 검출을 위한 알고리즘의 처리 속도를 높일 수 있다. 정지선 검출은 백색실선 정보를 이용하기 때문에 이미지 사이즈를 줄여도 정보의 손실이 적다. RGB 채널의 컬러 이미지를 단 채널의 그레일 스케일 이미지로 변환한다. 변환시킨 이미지를 가우시안 스무딩을 통해 잡음을 제거한다. 가우시안 스무딩은 마스크의 중앙값을 기준으로 외각 값이 작아지는 가우시안 마스크를 원영상과 컨볼루션함으로서 얻을 수 있다.
	(2)	정지선이 포함되는 차선 내부의 관심 영역을 위한 설정한다. <ul style="list-style-type: none"> Sliding window에서 투시변환에서 사용했던 것과 비슷하게 정지선이 포함되는 사다리꼴 모양의 차선내부의 관심 영역을 설정한다. 차선 내부는 1, 차선 외부는 0에 해당하는 픽셀값을 가지는 마스크를 만든다.
	(3)	마스킹을 통해 관심 영역을 추출하고 수평엣지 에지를 검출한다. <ul style="list-style-type: none"> 가우시안 이미지와 (2)에서 만든 마스크를 AND 연산 하여 정지선 검출을 위한 최종적인 관심 영역을 설정 한다. 관심 영역이 설정된 이미지에서 정지선을 검출한다. 정지선은 가로 방향이기 때문에 수평 에지 검출을 위한 마스크를 사용한다. 수평 에지 이미지에서 흐릿한 픽셀들을 제거하여 정지선을 명확하게 구분하기 위해 임계값을 적용한다. 영상 내 픽셀 값이 100 이상인 픽셀들을 대상으로 평균값을 구하여 임계값으로 설정한다.
	(4)	수평 엣지 이미지에서 정지선을 검출하고 차량에서 정지선까지의 거리를 추정한다. <ul style="list-style-type: none"> 수평 엣지 이미지에는 정지선을 포함하여 차선 같은 엣지로 검출된 모든 것들이 포함 된다. 영상의 픽셀값이 0~255로 바뀌는 지점을 상승 엣지, 255~0으로 바뀌는 지점을 하강 엣지로 하여 상승 엣지와 하강 엣지를 페어링한다.

	<ul style="list-style-type: none">- 정지선의 굵기가 일정하기 때문에 정지선의 엣지 페어링 길이보다 크거나 작은 엣지 페어링들은 제거하고 정지선을 검출한다.- 카메라의 화각과 차에 세팅된 환경을 고려하여 한 픽셀당 실제거리를 근사하여 정지선까지의 거리를 추정한다. <div><div>(5)</div><div>정지선이 감지되면 정지선 앞에 일정 거리를 유지하여 3초간 정차하고 다시 출발한다.</div></div> <ul style="list-style-type: none">- 정지선 앞에서 멈추기 전부터 속도를 서서히 주려 정지선과 일정 거리를 유지하며 3초간 정지한다.- 3초는 std_msgs/Header에서 time stamp에서 계산한 샘플링 시간 dt를 이용하여 측정한다.- 3초가 지나면 정지선 검출되기 이전의 속도로 다시 주행한다.									
<div>③ 기타 구현 기술과 관련된 내용 자유롭게 기술</div> <div>(구현 기술과 관련된 그림 포함)</div>	<ul style="list-style-type: none">- (3)에서 수평 엣지 마스크 <div><table><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-1</td></tr></table></div> <ul style="list-style-type: none">- (3)에서 수평 엣지 이미지 <div></div> <ul style="list-style-type: none">- (4)에서 엣지 페어링을 통한 정지선 검출 <div></div>	1	2	1	0	0	0	-1	-2	-1
1	2	1								
0	0	0								
-1	-2	-1								

수행 미션 (5번)	T자 수직주차 후 다시 출발 <ul style="list-style-type: none"> 후진으로 2대의 차량 사이에 수직으로 주차함. 주차 과정에서 양쪽 차량과 접촉사고가 나거나 후면 벽과 충돌하면 안됨. 3초 정차한 후 다시 빠져나와 주행트랙으로 복귀하여 계속 주행해야 함. 5초 이상 계속 주차하고 있으면 안됨. 	
① 사용하는 센서별 용도	카메라	영상처리를 통해 양쪽 차선의 위치를 파악하기 위해 사용
	초음파센서	주차 공간의 위치를 찾을 때 활용 후진할 때 뒤에 차량 혹은 벽면과의 거리 측정에 사용
	라이다	전방 장애물과의 거리 측정에 사용
	IMU	현재 차량의 속도와 Yaw값을 측정에 사용
② 구현할 기 능 요약, 그리 고 구체적 구 현방안	(1)	주차 구역이 되면 주차 구역과 일정거리를 유지하며 주행하다 주차구역을 찾는다.
		<ul style="list-style-type: none"> - 주차할 옆면과 라이다, 초음파 센서를 통해 거리를 측정하면서 일정 거리(3m)를 유지한다. - 차량 옆면과 주차구역 사이의 거리가 멀어지는 시점부터 주차 가능 공간 시작점이 된다. - IMU센서의 현재속도와 샘플링 시간으로 이동거리를 구하게 되고, 차량 옆면과 주차 구역 사이의 거리가 다시 일정거리(3m)를 유지하게 되는 시점이 주차 가능 공간의 마지막 점이 된다. - 주차 공간 시작점과 주차 공간 마지막 점 사이의 이동거리가 곧 주차 공간 구역의 길이이므로 주차 가능한지 불가능한지를 판단한다.
	(2)	조향각을 조절해가며 속도를 최소한으로 후진 주차를 진행한다.
		<ul style="list-style-type: none"> - 아래 그림에서 보면 주차가 되었을 때 차량 앞쪽의 위치(X점)을 원점으로 설정한다. - R1의 호의 거리는 왼쪽으로 조향각을 최대로 하고 차량이 전진하는 이동 거리이고 R2의 호는 오른쪽으로 조향각을 최대로 하고 차량이 후진하는 이동 거리이다. - R1과 R2의 반지름은 최소 회전 반경의 반지름이다. - 원점을 기준으로 R1, R2의 반지름을 이용하여 두 원의 중심 좌표는 두 원이 접하는 것을 이용하여 계산할 수 있고 각 호의 길이 즉 차량이 이동해야 하는 거리를 구할 수 있다. - 차량이 X점 즉 원점을 기준으로 $y=0$인 지점에 도착하면 조향각을 0도로 제어하고 후진한다. - 후진하다가 후면 초음파 센서를 통해 후면 벽과의 거리가 30cm일 때 정지한다.
	(3)	잘못 주차 시 조향각을 조절하며 전진과 후진을 반복한다.
		- 차량 옆면에 부착되어 있는 초음파 센서로 양쪽 차량과의 거리의 차이의 절댓값을

	<p>측정하고, IMU 센서로부터 차량의 yaw 값을 측정한다.</p> <ul style="list-style-type: none"> - 만일 양쪽 차량과의 거리의 차이의 절댓값이 제한된 값 이상이거나 yaw 값이 0이 아니면 재주차를 실시한다. - 재주차를 실시할 때는 5km/h의 전진과 함께 후진해서 들어온 조향값을 초기값으로 하여 0도로 천천히 조향한 이후에, 조향의 변화 방향을 유지하다가 차량이 원점을 기준으로 $x=0$이 되었을 때 yaw 값이 0이 될 수 있도록 조향의 변화 방향을 반대로 제어한다. - 원점 기준으로 $x=0$이며 yaw 값이 0이면, 후진을 실시하고 조향값 0을 유지하며 후면 초음파 센서를 통해 후면 벽과의 거리가 30cm일 때 정지한다. <p>(4) 주차를 마치고 주행 트랙으로 복귀한다.</p> <ul style="list-style-type: none"> - 3초는 std_msgs/Header에서 time stamp에서 계산한 샘플링 시간 dt를 이용하여 측정한다. - IMU를 통해 속도와 yaw값이 모두 0인 상황이 3초 이상 지속되면 (2)에서 계산한 각 구역에서의 이동 거리를 참고하여 제어를 반대로 진행하여 주차구역을 빠져나온다. - 주행 구역에 진입하기 이전처럼 카메라로부터 온 이미지에서 양쪽 차선의 위치를 파악하여 주행을 이어나간다.
<p>③ 기타 구현 기술과 관련된 내용 자유롭게 기술</p> <p>(구현 기술과 관련된 그림 포함)</p>	

수행 미션 (6번)	평행주차 <ul style="list-style-type: none"> 후진 또는 전진으로 트랙과 수평으로 놓인 주차구역에 주차함. 벽 또는 장애물과 충돌하면 안됨. 주차구역 앞쪽 벽에 붙여 놓은 AR코드를 이용하여 차량이 AR코드를 정면으로 바라보는 위치에 똑바로 정확히 주차해야 함. 	
① 사용하는 센서별 용도	카메라	AR 태그 인식할 때 사용
	초음파센서	주차 공간의 위치를 찾을 때 활용 후진할 때 뒤에 차량 혹은 벽면과의 거리 측정에 사용
	라이다	전방 장애물과의 거리 측정에 사용
	IMU	현재 차량의 속도와 Yaw값을 측정에 사용
② 구현할 기 능 요약, 그리고 구체적 구 현방안	(1)	주차 구역이 되면 주차 구역과 일정거리를 유지하며 주행하다 주차구역을 찾는다.
		<ul style="list-style-type: none"> - 주차할 옆면과 라이다, 초음파 센서를 통해 거리를 측정하면서 일정 거리(2m)를 유지한다. - 차량 옆면과 주차구역 사이의 거리가 멀어지는 시점부터 주차 가능 공간 시작점이 된다. - IMU센서의 현재속도와 샘플링 시간으로 이동거리를 구하게 되고, 차량 옆면과 주차 구역 사이의 거리가 다시 일정거리를 유지하게 되는 시점이 주차 가능 공간의 마지막 점이 된다. - 주차 공간 시작점과 주차 공간 마지막 점 사이의 이동거리가 곧 주차 공간 구역의 길이이므로 주차 가능한지 불가능한지를 판단한다.
	(2)	조향각을 조절해가며 속도를 최소한으로 후진 주차를 진행한다.
		<ul style="list-style-type: none"> - 아래 그림과 같이 직선 주행을 위해 조향각 0도로 제어해야 하는 P0~P1구간, 최소 회전 반경을 위해 조향각을 최대로 하여 시계방향, 반시계 방향으로 제어해야하는 P1~P2, P2~P3, P3~벽면 근처 구간으로 나누어진다. - 후진 주차 시작 위치와 후진 주차 목표 위치는 각 P0, P3로 나타낸다. - P1과 P2는 조향각이 바뀌는 점의 좌표이다. - 두 원의 중심 좌표는 두 원이 접하는 것을 이용하여 계산할 수 있고, P1, P2 또한 계산할 수 있다. - P0에서 P1까지의 거리, P1~P2와 P2~P3사이의 호의 길이를 계산하면 해당 구간에서 차량의 이동 거리를 구할 수 있다. - P3에 도착한 경우, 조향값을 0으로 맞춘 이후 라이다를 통해 전방에 있는 주차벽과의 제한된 거리까지 천천히 전진하다가 정지한다.
	(3)	잘못 주차 시 조향각을 조절하며 전진과 후진을 반복한다.
		- 카메라로 AR 태그를 통해 얻은 X,Y 좌표값과 Yaw 값을 활용한다. heading(차량의

방향과 주차벽의 가운데 지점의 각 = 90도에서 X/Y의 arctan 값과 Yaw의 합을 빼 값을 통해 얻은 값)으로 차량이 AR 코드를 정면으로 바라보는지 확인한다. (아래 그림을 참조)

- 정면으로 바라보지 않는다고 판단하면, 재주차 실행한다.
- 재주차를 실행할 때는, heading이 0보다 크다면 조향값을 왼쪽으로 조금씩 틀어주고 0보다 작다면 조금씩 오른쪽으로 틀어주며 heading이 0이 되면 조향을 0으로 유지하며 P3지점까지 천천히 후진한다.
- 이때 heading이 0이라면, 조향값을 0으로 맞추고 라이다를 통해 전방에 있는 주차벽과의 제한된 거리까지 천천히 전진하다가 정지한다.
- 만일 후진으로 P3지점까지 다시 왔는데 주차벽을 정면으로 바라보고 있지 않다면, P1을 목표 지점으로 하여 (2)과정을 반대로 전진을 진행한다.
- 그 이후 (2)과정을 재실행한다.

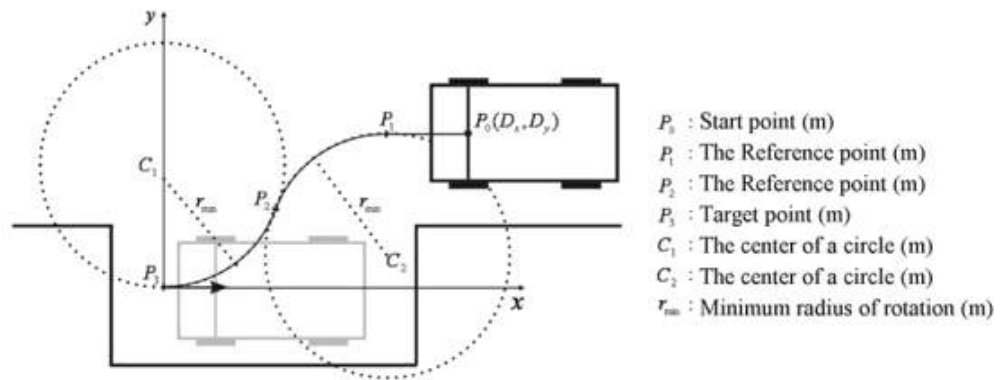
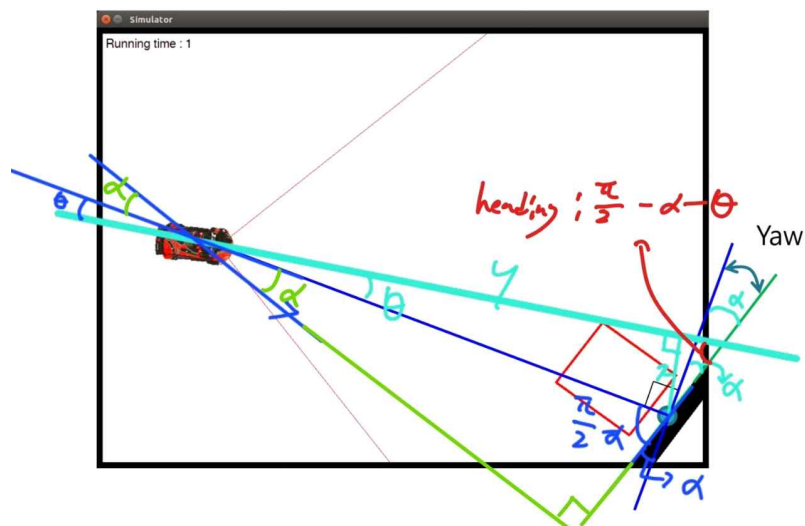


Fig. 2-11 Ideal path for a parallel parking.



③ 기타 구현
기술과 관련된
내용 자유롭게
기술

(구현 기술과
관련된 그림
포함)