제17회 임베디드SW경진대회
개발완료보고서 [자율주행 모형자동차]

0. 작성 시 주의사항

※아래의 작성 양식(제출분량, 폰트, 크기, 줄 간격 등)을 미준수 시 서류 평가의 감점요인됨

※ 제출 분량 : A4 용지 상세내용 포함 30 page 이내
※ 작성 양식 (폰트 : 맑은 고딕 / 폰트 크기 : 10pt / 자간 : 0% / 장평 : 100% / 줄 간격 : 130%)
※ 제출 포맷 : pdf

1. 팀 정보

팀명	에이스 (SMW Ace)	팀장	남창호
팀원	김보경	팀원	김윤수
팀원	강이경	팀원	김남우

2. 개발완료보고서

1. 개요

1.1. 작품 개요

카메라를 통한 영상인식으로 차선과 신호등과 같은 외부상황을 인지하고 적외선 센서를 통해 주변 물체와의 거리를 확인하며 스스로 주행하는 모형차이다. 내리막, 터널, 추월차로 등의 약 10개의 미션 코스를 소프트웨어로 구현한 알고리즘을 바탕으로 모형차가 자율적으로 판단하며 통과해, 최종 목적지에 도달하게 된다.

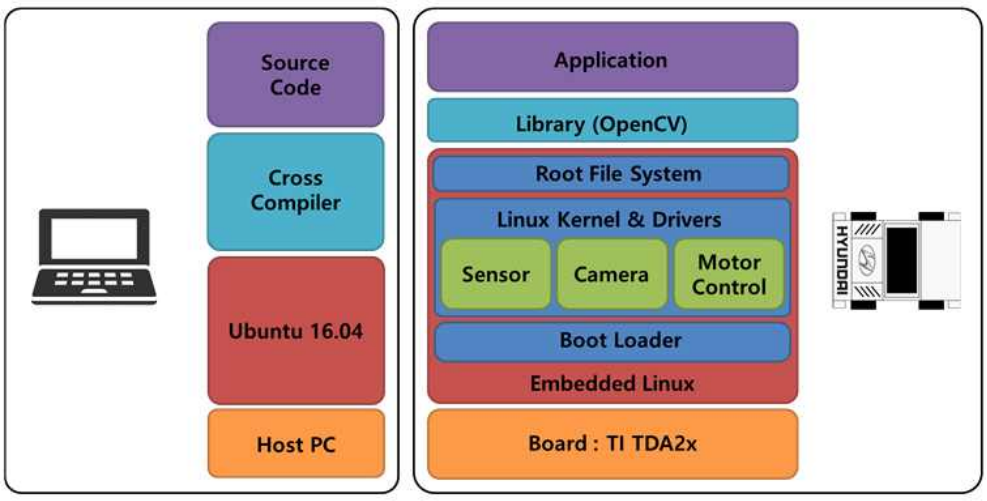
1.2. 개발 목표

자율주행 모형자동차가 코스 이탈 등의 사고가 발생하지 않으면서 최대한 안전하고 신속하게 미션 코스를 통과하여 결승지점에 도착하는 것이 개발 목표이다.

개발 과정에서 자율주행의 기본 요소인 상황 인지 및 판단과 그에 따른 모션 제어를 모형차 구현을 통해 해결해보면서 이를 바탕으로 실제 차량의 5단계 자율주행 구현 시에 마주할 수 있는 다양한 문제들에 대해 생각해보고자 한다. 또한 자율주행 산업발전에 기여할 수 있는 인재가 되기 위해 소프트웨어뿐만 아니라 현재 개발 중인 NVIDIA 플랫폼을 기반으로 한 자율주행 SW구현 프로젝트 속에서도 신뢰성과 효율성이 보장된 자율주행 시스템 개발을 목표로 삼는다.

2. 개발 환경 설명

2.1. Software 구성



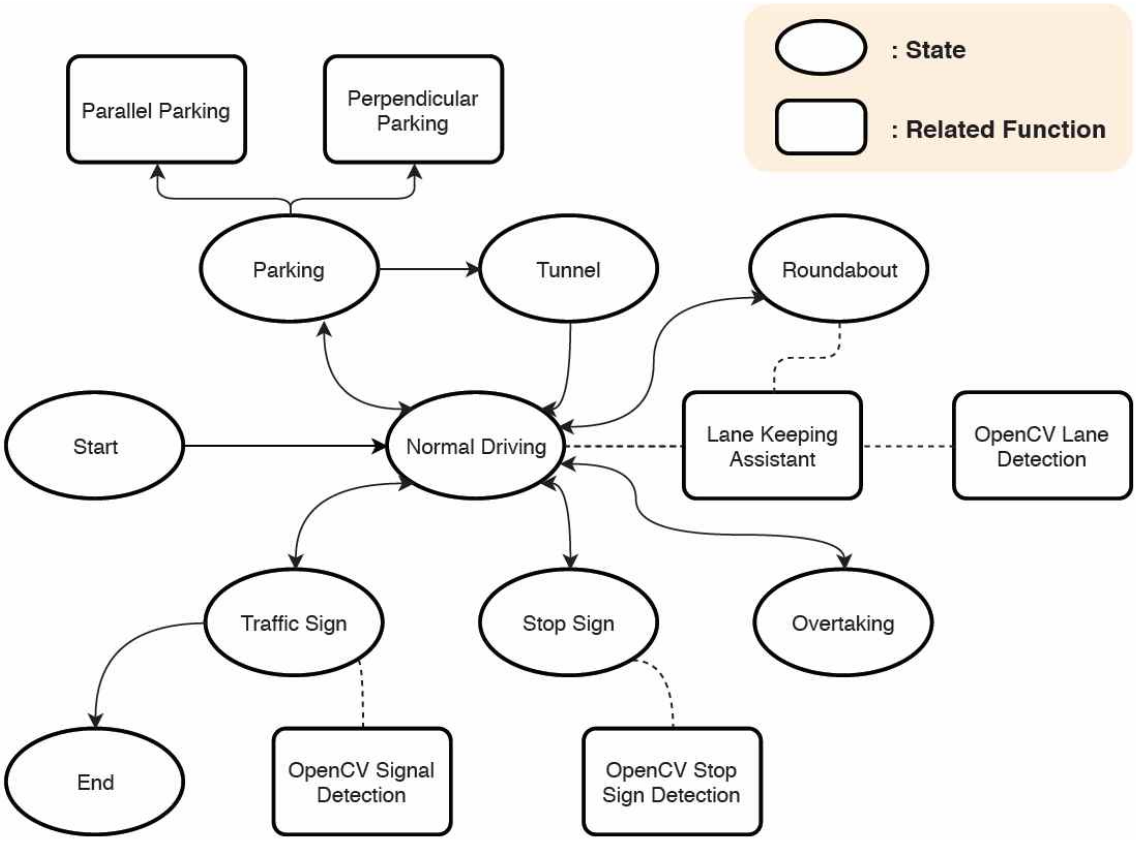
개발 Software 구성

프로젝트에서 사용되고 개발된 전체 Software 구조는 위와 같다.

코드를 작성하고 실행파일을 만들어 주는 Host PC의 OS로는 Ubuntu 16.04가 사용되었다. 모형자동차 제어 코드를 .c 파일로 작성하고 ARM Architecture에서 실행될 수 있는 ELF 파일을 생성해주기 위해 GCC Cross Compiler가 사용되었다.

모형자동차의 임베디드 보드는 TI TDA2x가 사용되었다. 그 위에 OS로 Embedded Linux가 사용되었다. Linux Kernel의 Device Driver들이 모형자동차에 장착된 센서와 모터 등의 하드웨어를 직접적으로 제어하는 기능을 한다. 영상처리 알고리즘에 활용되는 Library로 OpenCV가 활동되었다. 그리고 Host PC에서 작성된 자율주행 로직 코드를 Application 프로그램으로서 실행되어 모형자동차가 원하는 대로 동작하게 된다.

2.2. **Software 설계도(흐름도 및 클래스 다이어그램 등 (개발언어에 따라 선택))**



Software 설계도

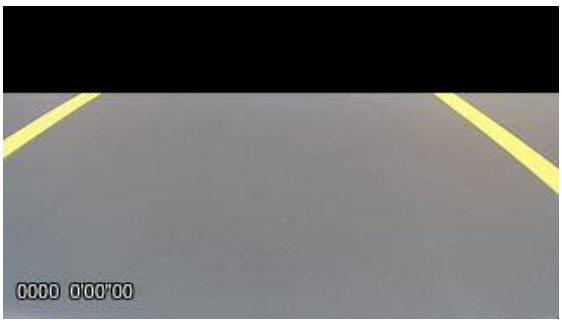
대회 규정에 명시된 각 미션 코스를 통과하는 자율주행 소프트웨어를 개발하기 위해 각 코스별로 State로 구분하여 설계하였다.

State의 구성으로는 Start, Parking, Tunnel, Roundabout, Traffic Sign, Stop Sign, Overtaking, End로 코스별로 나뉘었다. 모형자동차는 Start state에서 처음 대기 하였다가 출발하게 되면 Normal Driving state로 넘어가게 된다. 이후 대부분의 state 변화는 Normal Driving 상태에서 넘어가게 된다. 센서 및 카메라로 인해 특정 상황이 인지가 되면, 각 코스(State)진입 조건이 충족되면 해당 State로 넘어가고 그 state에 해당하는 로직이 수행된다. state 상태에서 모든 동작이 수행되고 완료 조건 충족 시 다시 Normal Driving state로 돌아와 일반 주행을 시작한다.

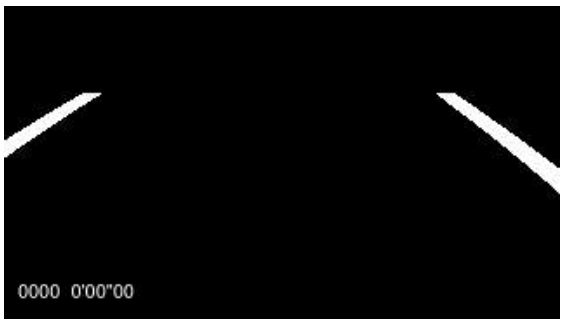
2.3. Software 기능 (알고리즘 설명 포함)

2.3.1. 일반 주행(직선, 코너 주행)

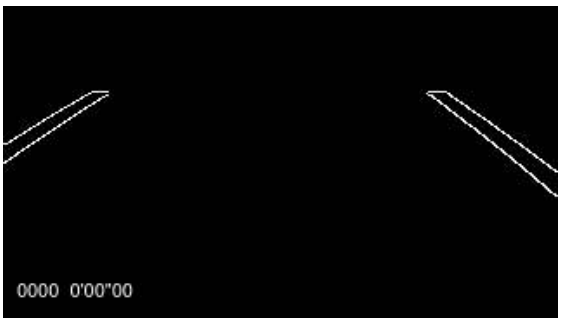
1) 차선 인식 알고리즘



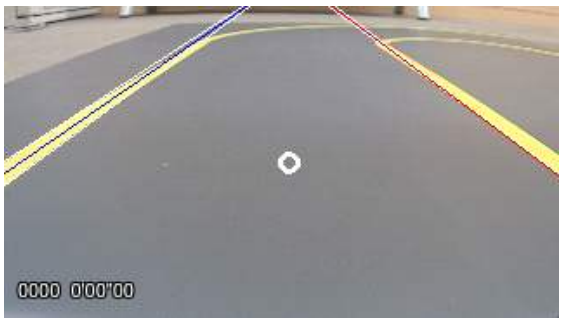
ROI 적용 이미지



HSV 차선 추출 이미지



Canny edge 추출 이미지



Hough line, 좌우 대표 라인(평균), 소실점 추출

① 카메라로 입력되는 이미지에서 차선 트랙에 해당하는 부분만 ROI로 잘라내어 실시간 이미지 연산량을 줄이고, 배경 물체에 라인이 잘 못 인식될 수 있는 가능성을 줄인다.

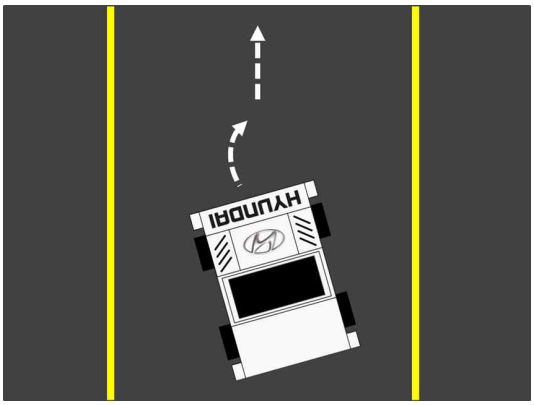
② 노란색 차선을 추출하기 위해 HSV 이미지에서 inRange를 이용해 이미지 필터링한다. 차선 부분만 흰색(255), 그 외의 부분은 검은색(0)으로 변환된다.

③ threshold가 적용된 이미지에 Canny edge를 추출한다.

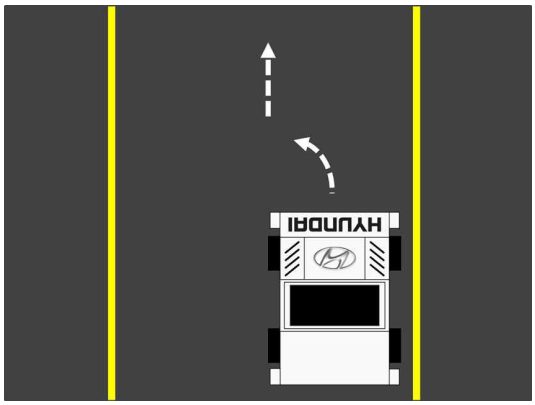
④ edge가 추출된 이미지에서 Hough line을 추출한다.

⑤ 추출된 선들의 rho, theta값들의 평균을 구하여 차선의 대표하는 선 두 개(직선 구간에서 양쪽 차선)를 얻는다.

2) 직선 주행 알고리즘



차량 orientation 보정

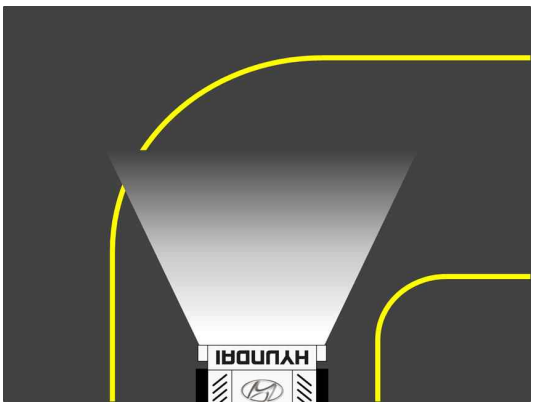


직선 차선 내 차량 위치 치우침

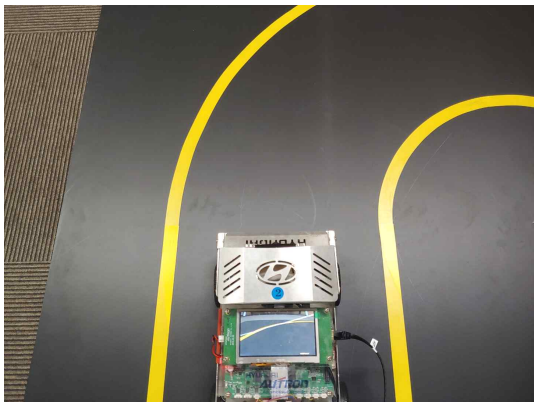
① 최종적으로 추출된 양쪽 차선의 교점(소실점)을 구하여 직선 주행 상황에서의 방향(orientation)을 보정한다.

② 차량의 orientation은 차선과 평행하지만 차선 내에서 한쪽으로 치우친 상태를 보정하기 위해 라인의 y절편 값을 계산하여 차선 중간지점으로 회귀할 수 있도록 한다.

3) 곡선 주행 알고리즘



커브 진입 시 모형차 카메라 화각



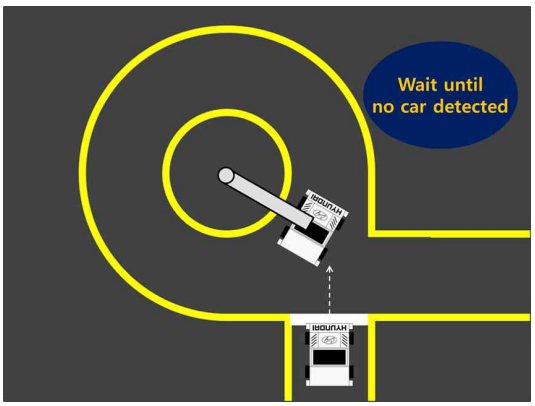
실제 커브 진입 시 인식되는 영상 이미지

① 곡선 주행 구간에서는 양쪽 차선 중 하나만 추출된다. 추출된 한쪽 차선의 기울기 값을 기본으로 바퀴 조향각을 제어한다.

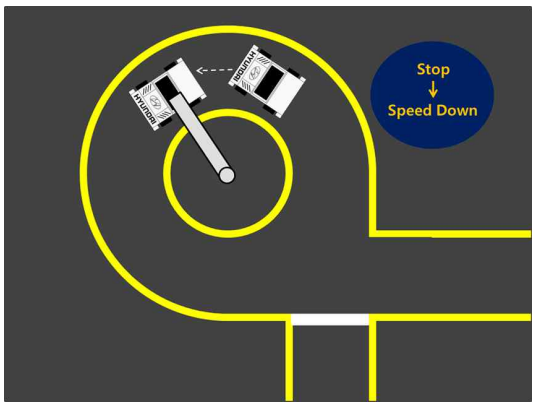
② y절편을 이용한 조향각 보정으로 곡선 구간 내에서 차선과의 간격을 유지하고 부드러운 곡선 주행이 가능하도록 한다.

※ 위의 커브 구간 진입 상황의 그림처럼 차가 커브구간에 완전히 진입하지 못했음에도 불구하고, 카메라는 이미 하나의 선만 인식하게 된다. 이때 일정 값 이상의 y절편 값 조건을 적용하여 완전히 진입 후 조향 동작을 할 수 있게 한다.

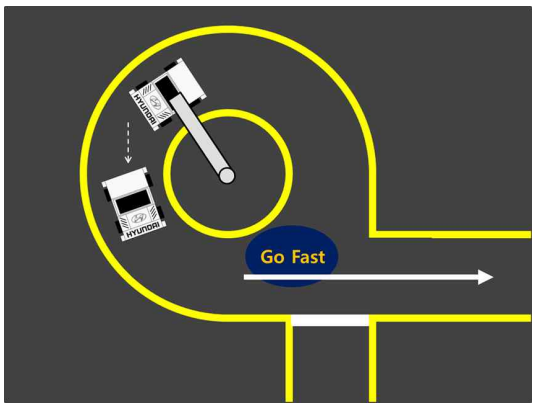
2.3.2. 회전 교차로



정지선에서 차가 감지 안 되면 진입



전방에 차량 감지될 시



후방에 차량 감지될 시

1) 라인 센서를 통해 흰색 정지선이 감지되면 차가 정지하여 Roundabout(회전 교차로) state로 인식 된다.

2) 전방에 지나가는 차를 거리 센서를 이용해 일정시간 감지 후 차가 없다면 교차로 안으로 진입하게 된다.

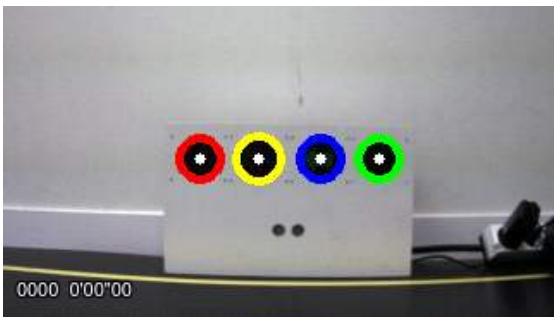
3) 전방의 거리 센서에 차량이 감지되면 잠시 정지했다가 이전보다 낮은 속도로 다시 출발한다.

교차로 내의 차보다 속도가 빨라 당장 앞질러 갈 수 없다고 해도, 이 과정에서 자연스럽게 전방 차량의 속도와 일정하게 되어 Cruise Control 효과가 나게 된다. 따라서 안전하고 신속하게 교차로를 통과할 수 있다.

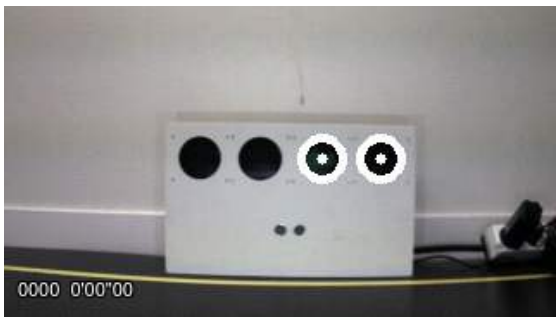
4) 만일 후방 거리 센서에 교차로 내의 차량이 감지되면 전방에는 차량이 존재하지 않는다는 의미 이므로 속도를 올려 교차로를 빠르게 빠져나가도록 한다.

2.3.3. 신호등 분기점

1) 신호등 원 검출 알고리즘



Hough circle, 신호등 원 검출



좌·우회전 신호등 위치 확정

① 라인 센서를 통해 흰색 정지선이 감지되면 차가 정지하여 TRAFFICLIGHT(신호등) state로 인식 된다.

② 신호등 부근에 ROI를 설정한다.

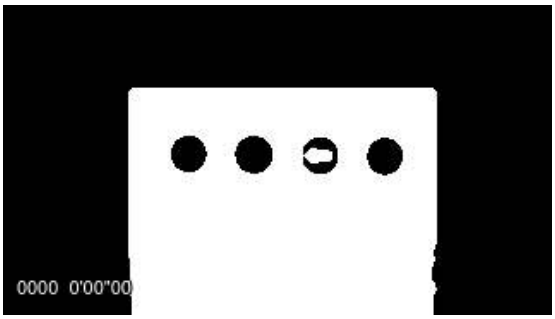
③ hough circle 함수를 이용해 신호등 원을 검출하고, x값을 기준으로 정렬하여 좌·우회전 신호등의 좌표를 구한다.

④ 이 좌표 값을 이전 프레임에서 구한 좌표 값과 비교하여 차를 구한다.

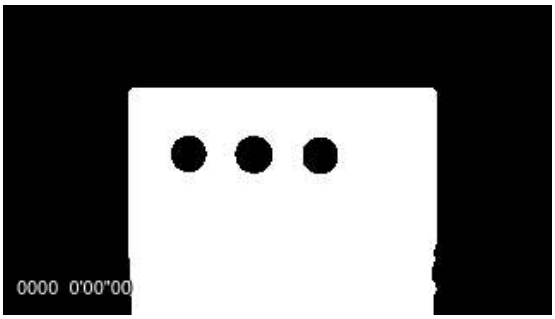
⑤ 차의 값이 일정 값 이상이면 에러 값이라고 판단하고, 일정 값 이하이면 count 값을 증가시킨다. 이 과정을 반복해 count 값이 n이 되면 좌·우회전 신호등 원의 위치를 확정한다.

⑥ 좌·우회전 신호등의 위치가 확정되고 나면 다음 프레임부터는 이 과정을 건너뛰고, 신호등 on/off 여부만 확인함으로써 속도를 개선한다.

2) 신호등 ON/OFF 판단 및 조향 알고리즘



Threshold 변환, 좌회전 신호 ON



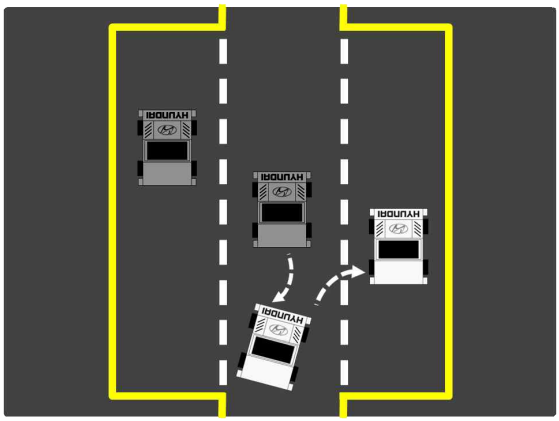
Threshold 변환, 우회전 신호 ON

① 일정한 Threshold 값을 기준으로 영상을 이진화한다.

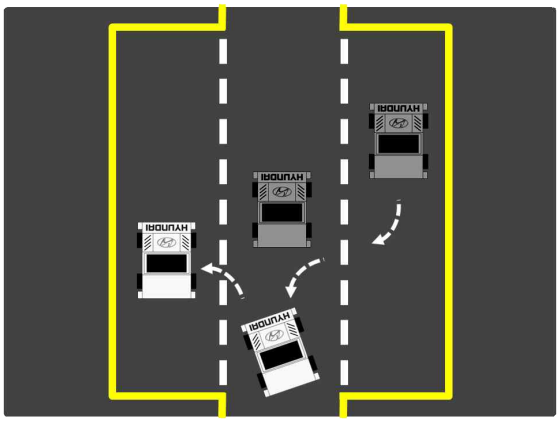
② 앞서 구한 좌·우회전 신호등 원의 중심부 주변 픽셀을 검사해 흰색 값(255)이 50%이상이면 불이 들어왔다고 판단하고, 조향한다.

③ 그렇지 않으면 stop 상태를 유지한다.

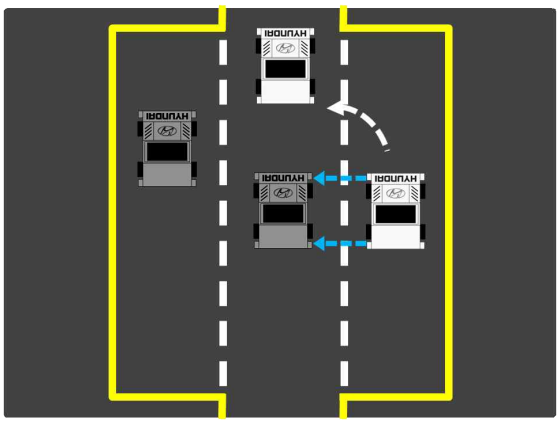
2.3.4. **추월구간**



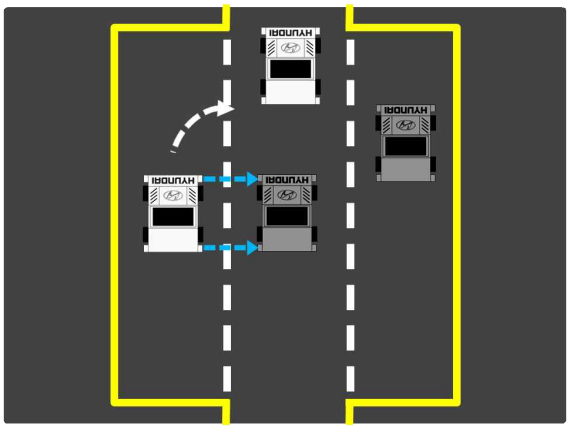
전방 센서 인식 후,
우측 차선 변경



전방 센서 인식 후,
좌측 차선 변경



우측 차선 주행 중,
좌측 후방 센서 인식 후 중앙 차선 복귀

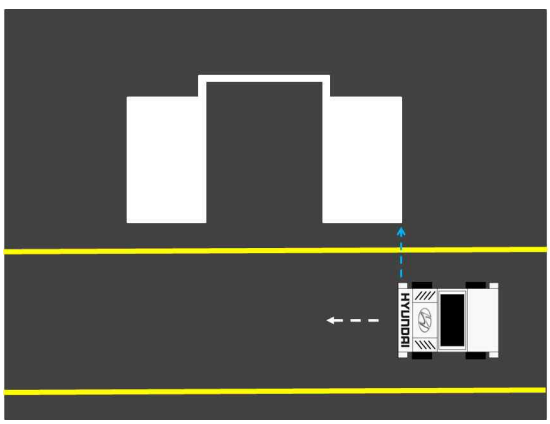


좌측 차선 주행 중,
우측 전방 센서 인식 후 중앙 차선 복귀

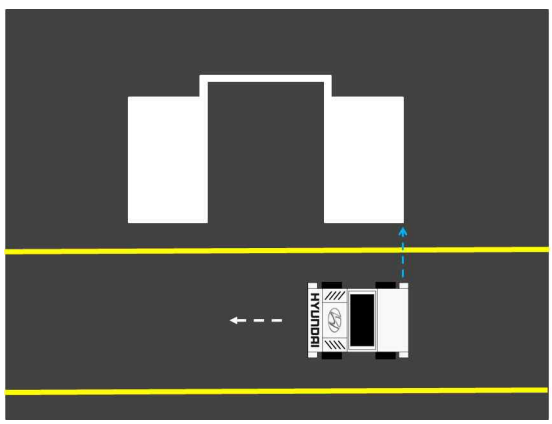
1) 전방 거리센서가 감지되면 OVERTAKING(추월) state로 전환한다.
2) 뒤로 후진 후 오른쪽 차선으로 진입하여 주행한다.
3) 전방 거리센서가 감지(우측 차선에 차량 존재)되면 후진하여 중앙 차선으로 복귀한 후 왼쪽 차선으로 진입한다.
4) 차선을 따라 주행하면서 측면 거리센서(왼쪽 차선일 경우: 우측 전후방 센서, 오른쪽 차선일 경우: 좌측 전후방 센서)가 감지되면 다시 중앙 차선으로 복귀한다.

2.3.5. **주차 구간**

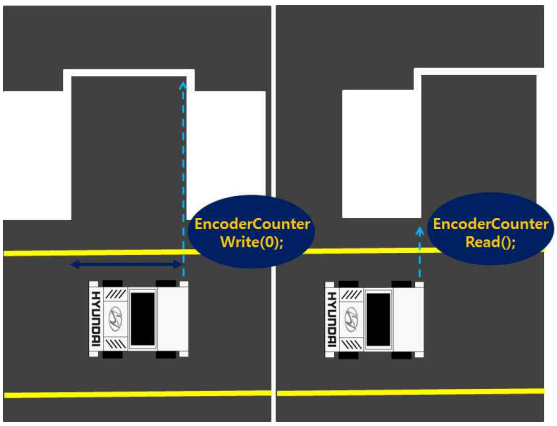
1) 수직/평행 주차 구간 판단 알고리즘



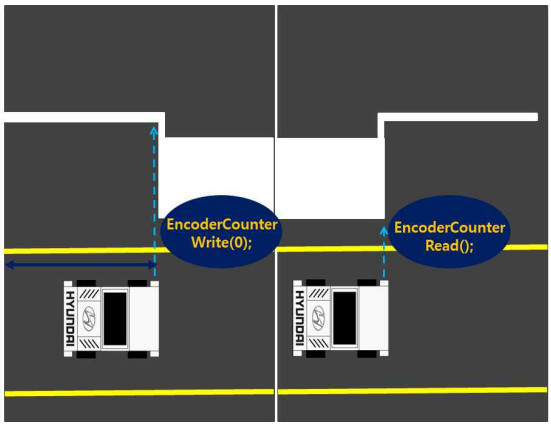
2번 센서를 통하여 장애물을 인식



3번 센서를 통하여 주차모드임을 인식



엔코더카운터를 이용하여 수직주차 인식



엔코더카운터를 이용하여 수평주차 인식

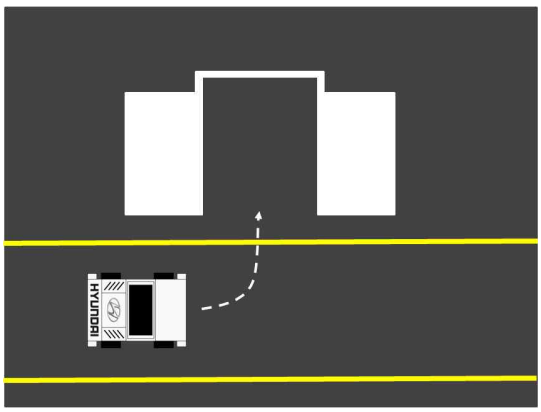
① 우측 전방 센서에서 구조물을 감지를 하면 주차 혹은 터널 모드인지 체크하기 시작한다.

② 우측 후방 센서가 구조물이 감지되기 전까지 좌측 전방 센서가 터널 구조물을 인식을 하지 않으면 주차모드로 들어간다.

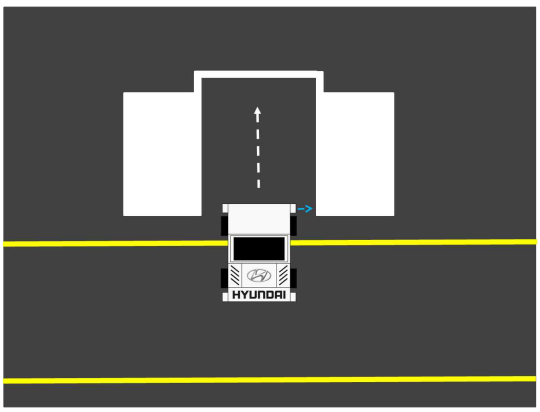
③ 그 이후 우측 후방 센서의 값이 특정 값 이하(주차구간)로 이하가 될 경우 주차구간에 들어왔음을 인식한다. 그 즉시 엔코더 카운팅을 시작한다.

④ 우측 후방 센서를 통해 주차구간이 끝났음을 인식할 때 엔코더 카운터 값을 읽어낸다. 그 값을 사용하여 수직, 평행 주차모드를 구분한다.

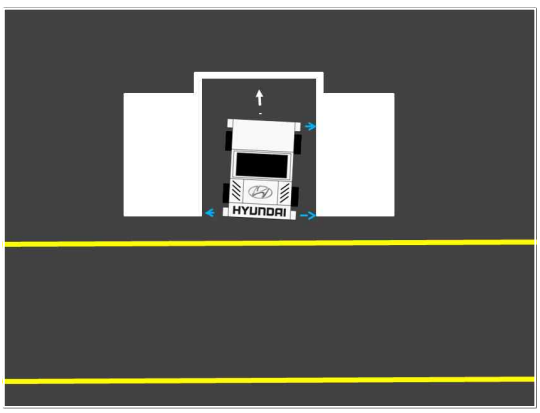
2) **수직 주차 알고리즘**



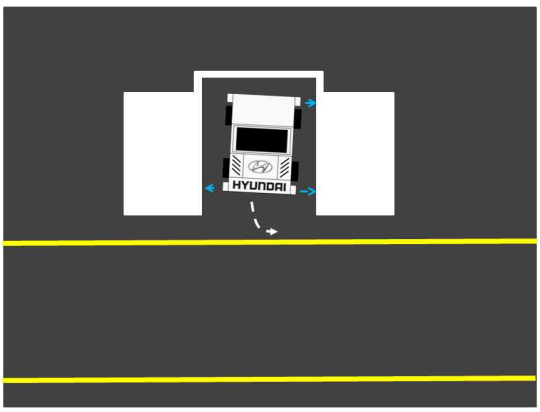
주차 공간으로 조향 제어하여 후진



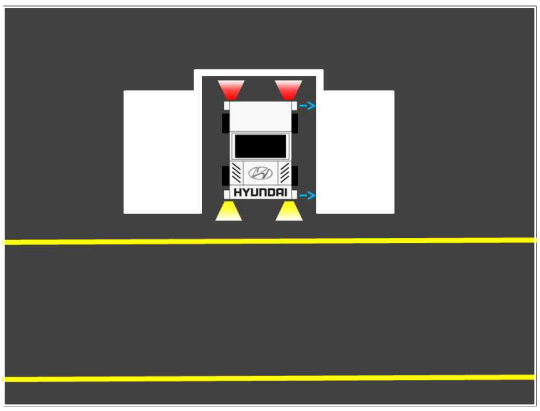
좌측 후방 센서 인식하면,
나란하게 조향 제어하면서 후진



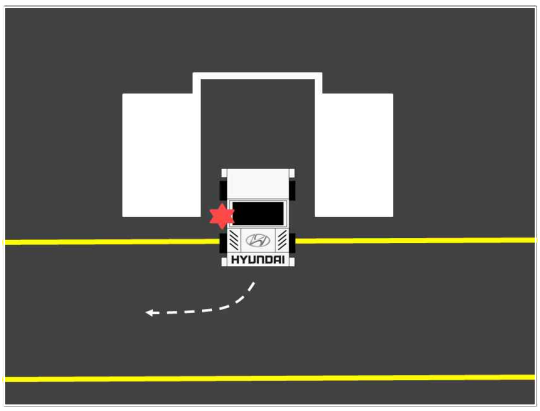
센서 비교하면서 조향제어



나란하지 않고 후진할 공간이 없으면,
조향 제어하면서 직진



측면 센서를 사용해서 자세 조정, 주차 공간 내부로 진입 완료하면 부저 ON



측면의 센서 상하단부 사이에 도달했을 때 조향 제어하면서 우회전

① 주차 구간으로 진입할 수 있도록 조향각 제어하며 후진한다.

② 좌측 후방 센서가 주차구간을 인식하면 조향각을 정방향으로 제어 후 후진한다.

③ 우측 전방 센서까지 주차 구간으로 들어오면 측면 전후방 센서를 비교하여 모형차가 주차 공간에 나란하도록 자세 조정하기 위해 조향각 제어를 하면서 후진한다. 센서는 4cm이하의 근접한 거리에서는 신뢰성을 잃어버린다. 따라서 좌우 전방 센서를 비교하여 4cm 이상의 측면의 센서값을 도출해낸다. 그리고 신뢰성 있는 측면의 전후방 센서를 사용하여 비교한다.

③-1 모형차가 주차 구간에 나란할 경우 그대로 후진하여 주차구간까지 안전하게 진입하고 부저를 울려 주차를 성공적으로 완료했음을 알린다.

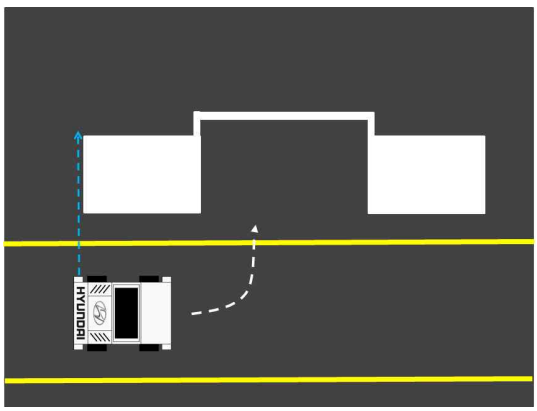
③-2 모형차가 나란하지 않고 주차구간 충돌 위험 영역까지 후진을 했다면 멈춘다. 그리고 측면 전후방 센서를 비교하여 자세 조정을 위해 조향각을 제어하고 전진한다.

③-3 모형차가 나란해질 때까지 후진, 전진을 반복한다.

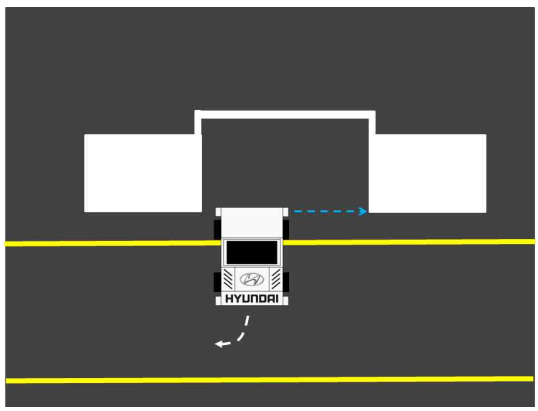
④ 주차 완료 후 직진한다. 전후방 센서값 인식 직전, 직후 조향각을 제어할 경우 주차구간과 충돌하거나 혹은 주행 라인을 벗어날 가능성이 있다. 따라서 안전하게 좁은 수직 주차구간을 벗어나기 위해서 우측 전방 센서가 주차구간을 벗어나자마자 적절한 엔코더 값을 주어 전후방 센서 중간 부근까지 도달할 때까지만 직진하도록 한다.

⑤ 그 이후 우회전을 하도록 조향각 제어를 하여 안전하게 차선 안으로 진입하도록 한다.

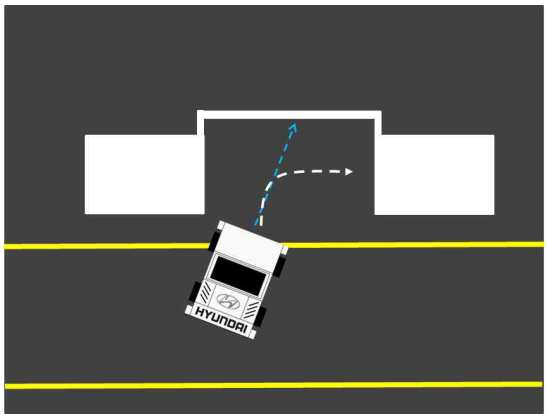
3) 평행 주차 알고리즘



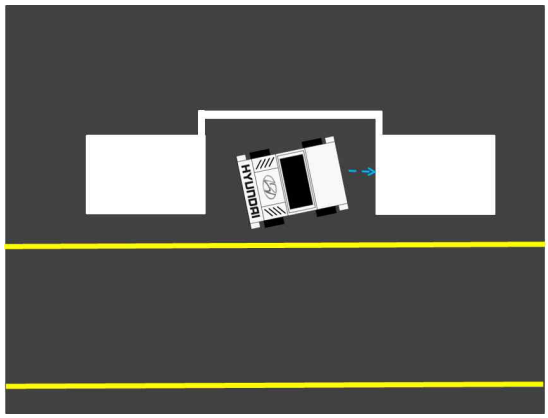
측면센서 인식 후 우측으로 조향하며 후진



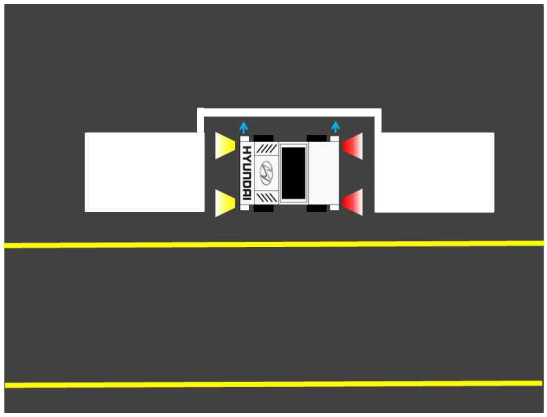
좌측 후방센서가 인식되면 대각선 자세로 진입을 위한 우측 조향 및 전진



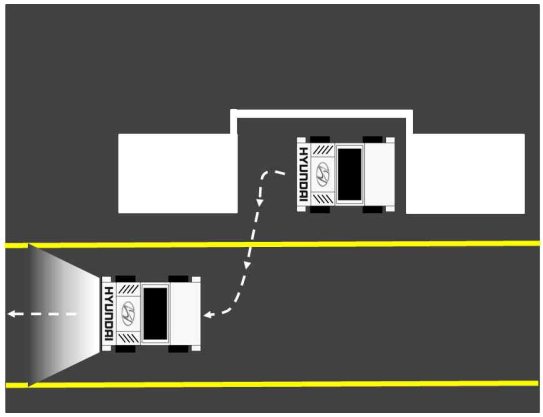
후방 센서 감지하며 주차공간으로 진입



후방 센서가 우측 벽면으로부터 일정 거리 이하를 감지할 때까지 후진



주차공간과 측면 센서의 평행을 확인하면 주차 완료 후 부저 울림



위치 제어를 통해 주차 구역에서 빠져나옴

① 주차 알고리즘을 통하여 평행 주차 구간임을 인식하고 후진으로 주차구역으로 진입한다.

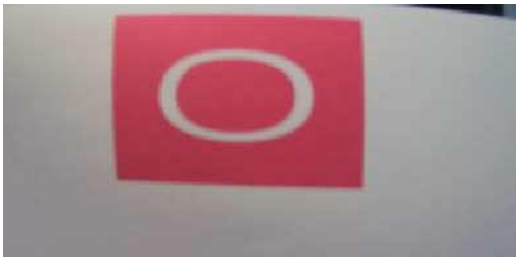
② 좌측 후방 거리 센서가 우측 벽면을 감지할 때 까지 우측으로 꺾인 조향각을 유지하며 후진한다.

③ 좌측으로 조향각을 꺾은 후 후방 거리 센서가 우측 벽면을 일정 거리 이하로 감지할 때까지 완전히 주차구역으로 들어선다.

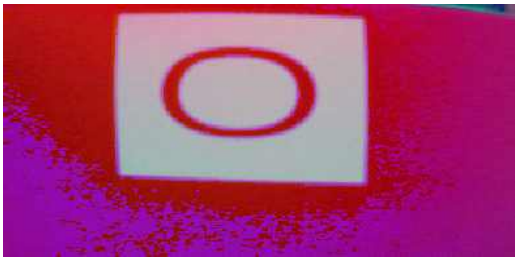
④ 우측으로 조향을 한 후 전진을 하여 우측 전, 후방 센서 비교를 통해 모형차의 자세가 평행함이 인지되면 주차가 완료 되며 부저를 울린다.

⑤ encoder를 이용한 위치 제어를 통해 주차구역을 완전히 빠져나온 후 Normal driving state로 진입하여 주행한다.

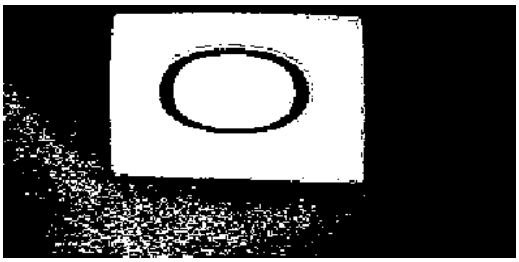
2.3.6. **일시정지**



원본 이미지



HSV 적용 이미지



이진화 적용 이미지

ROI 적용 이미지

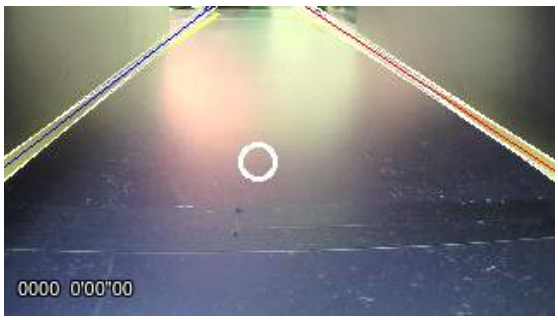
1) 일반 주행 중(Normal driving)에 일정 시간마다 저장된 이미지를 원하는 색 추출하기 위해 HSV변환을 한다.
2) 변환된 이미지를 실험을 통해 도출된 red색상을 최대한 인식하는 HSV값을 찾아냈다. 그 값을 inRange 함수에 매개변수로 넣어 이진화된 이미지를 얻어낸다. 최대한 환경에 영향을 받지 않기 위해 채도(S), 명도(V)를 최솟값 0에서 최댓값 255를 맞춰놓은 상태에서 red가 잘 추출되는 색상(H)의 범위를 알아내도록 했다.
3) 이진화 된 이미지에서 픽셀 값을 읽어내고 적정한 픽셀 값 이상에 도달하면 일시정지를 한다.
4) 지속적으로 픽셀 값을 읽어내고 표지판이 카메라 영역에서 없을 수준으로 픽셀 값이 도달할 경우 약간의 delay(1.5초)를 준 다음 출발한다. 바로 출발할 경우 표지판과 부딪힐 위험이 있기 때문이다.
5) 일시정지 표지판을 인식하고 동작을 완수한 이후부터는 일시정지 표지판을 검출하는 함수를 더 이상 호출하지 않도록 한다.

2.3.7. **터널**

터널 내에서 인식된 영상 이미지

ROI 추출 및 HSV Filter 적용

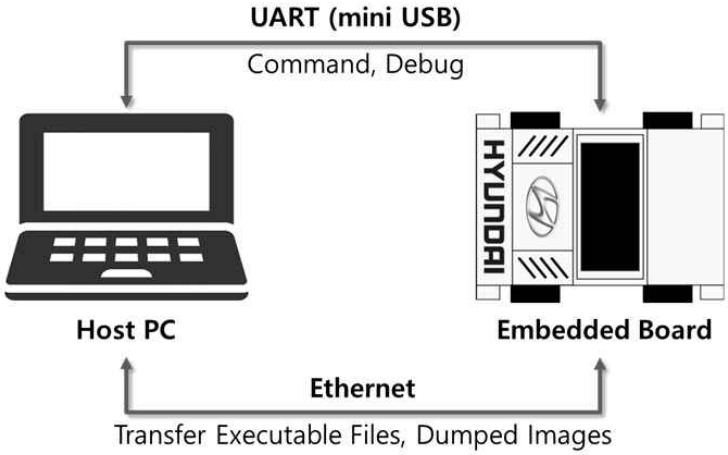
Canny edge 추출



Hough transform 적용하여 line 추출

1) 진입조건은 좌우측 전방 거리 센서(2, 6번 센서)가 터널 구조물 벽면을 인지하면 터널 진입 상태로 인지된다.※ 우측 전방 센서(2번 센서)가 먼저 감지되면 주차 상태로 들어가게 된다. 이 때 추가로 좌측 전방 센서(6번 센서)가 감지되면 터널 상태로 진입되도록 예외처리 하였다.
2) 터널에 진입하게 되면 조도를 확보하기 위해 전방의 LED Light가 켜진다.
3) 터널 내에서 차선 인식 및 주행 방식은 Normal driving state와 동일하다. LED Light로 인한 노이즈 발생을 방지하기 위해 터널과 외부에서 모두 적용되는 HSV Filter 값을 설정하여 터널 내에서도 바깥과 동일한 차선 인식이 가능하다.

2.3.8. **프로그램 사용법 (Interface)**



Interface 환경

Host PC와 모형자동차의 Embedded Board 사이의 Interface는 UART와 Ethernet 통신을 이용하였다. UART는 Embedded Linux 환경에 접근하여 명령어를 실행하고, 프로그램이 실행되면서 출력하는 데이터를 모니터링하기 위해 사용했다. Ethernet은 Host PC에서 Cross Compile된 ELF 파일을 Board로 전송하거나, 모형차의 카메라로 Dump한 image를 Host PC로 전송하기 위해 사용되었다.

2.4. 개발환경 (언어, Tool, 사용시스템 등)

2.4.1. 개발환경

1) 운영체제 : Linux (Ubuntu 16.04)
2) 터미널 프로그램 : Minicom
3) 코드 에디터 : Visual Studio Code
4) 프로젝트 관리 : Github

2.4.2. 통신

1) Ethernet 통신
   ① SSH 프로토콜을 이용하여 Local PC에서 크로스 컴파일 된 ELF 파일을 보드로 전송하고, 보드에서 dump한 이미지 파일을 전송하는데 사용되었다.
2) Serial 통신
   ① UART 프로토콜을 이용하여 프로그램에서 printf로 센서 데이터 등을 문자를 출력하여 디버깅하는데 유용하게 사용되었다.
   ② Local PC에서 minicom 터미널에 입력한 명령어를 보드의 Embedded Linux로 전달하여 프로그램을 실행하는 등의 명령을 실행하였다.

2.4.3. 언어

1) C
   ① 모형자동차는 Linux application 프로그램이 실행되어 제어된다. 이 제어 application은 C 언어 코드로 작성 되었다.
2) C++
   ① OpenCV를 사용하기 위해 C++가 사용된다. exam_cv.cpp 파일에서 OpenCV를 이용한 영상처리 함수들을 C++기반으로 작성하고 메인 프로세스에서 이 함수들을 가져다 사용하게 된다.

3. 개발 프로그램 설명

3.1. 파일 구성

3.1.1. **exam_cv.cpp**

1) 카메라를 통해 streaming 되는 이미지들을 OpenCV를 이용해 원하는 데이터를 추출하기 위한 함수가 구현되어있는 파일이다.

2) 구성 함수
OpenCV_hough_transform, OpenCV_color_detection, OpenCV_signal_detection, OpenCV_color_detection_stopsign

3.1.2. **car_lib.c**

1) 하드웨어를 제어하기 위한 API 및 제어 시퀀스 함수가 모여 있는 파일이다.

2) 구성 함수
CarControlInit, CarSetInit, CarLight_Write, Alarm_Write, Winker_Write, SpeedControlOnOff_Read, void SpeedControlOnOff_Write, DesireSpeed_Read, DesireSpeed_Write, SpeedPIDProportional_Read, SpeedPIDProportional_Write, SpeedPIDIntegral_Read, SpeedPIDIntegral_Write, DesireEncoderCount_Write, EncoderCounter_Read, EncoderCounter_Write, SteeringServoControl_Read, SteeringServoControl_Write, CameraXServoControl_Read, CameraXServoControl_Write, CameraYServoControl_Read, CameraYServoControl_Write, LineSensor_Read, DistanceSensor, perpendicularParking, parellelParking

3.1.3. **main.c**

1) 미션 코스별 state 기준으로 함수들이 구성되어있다. 전체 Software 설계에 따라 자율주행 로직을 실행시켜주는 파일이다.

2) 구성 함수
state_Start, state_NormalDriving, state_Parking, state_StopSign, state_Roundabout, state_Tunnel, state_Overtaking, state_TrafficLight, state_End

3.1.4. **함수별 기능**

1) **state_Start**
프로그램이 실행 될 때 가지는 초기 state 함수이다. IR 센서를 통한 출발 신호를 대기했다가 센서가 감지되면 Normal Driving state로 바뀌며 출발한다.

2) **state_NormalDriving**
인식된 차선의 데이터를 이용하여 직선과 커브 주행을 수행한다. LaneKeepAssistant 함수를 통해 steering 조향 제어가 된다. 이 state를 통해서 다른 미션 코스의 state로 빠져나가게 된다.

3) **state_Parking**
Normal Driving state에서 주행을 하다가 주차 구조물이 거리 센서에 감지가되면 Parking state로 변환 되어 state_Parking 함수가 실행된다. 이 때 수직, 평행 주차를 구분하기 위한 함수가 호출되며 그 반환값에 따라 perpendicularParking 또는 parallelParking 함수가 실행되어 수직/평행 주차 시퀀스가 실행된다.

4) **state_StopSign**
일시 정지 표지판이 카메라 상에 감지될 때 Stop Sign state로 변환되어 호출되는 함수이다. state_StopSign 함수는 일시 정지 표지판이 감지되는 동안 모형차가 정지 상태를 유지하도록 한다.

5) **state_Roundabout**
라인 센서를 통해 첫 번째로 일시 정지선이 감지되면 Roundabout state로 들어간다. state_Roundabout 함수가 실행되면 거리 센서로 지나가는 차량이 없는 지 살핀 후 교차로로 진입하도록 한다. 그리고 교차로 내에서 전방 및 후방에 차량이 감지되는 여부에 따라 속도를 조정하여 신속히 교차로를 빠져나올 수 있도록 한다.

6) **state_Tunnel**
우측 거리 센서에 장애물이 감지되면 실행되는 Parking state와 달리 좌우 양쪽 거리 센서가

감지되면 state_Tunnel 함수가 실행된다. 터널에 진입하면 거리 센서로 감지되는 좌우 센서 값의 오차를 기준으로 steering 제어를 하며 주행한다.

7) **state_Overtaking**
거리 센서로 전방에 차량이 감지되면 Overtaking state가 된다. state_Overtaking 함수는 오른쪽 혹은 왼쪽 차선으로 이동 후 거리 센서에 감지되는 차량을 기준으로 차선 이동 동작을 수행하여 코스를 빠져나오도록 한다.

8) **state_TrafficLight, state_End**
라인 센서가 두 번째로 일시 정지선을 인식하면 Traffic Light state로 들어간다. state_TrafficLight 함수는 전방에 신호등 표시를 감지하여 대기/좌회전/우회전을 수행하도록 한다. 그리고 초록색 불이 들어와 좌/우회전을 하여 최종 정지선을 만나게 되면 state_End로 들어가 프로그램을 종료하게 된다.

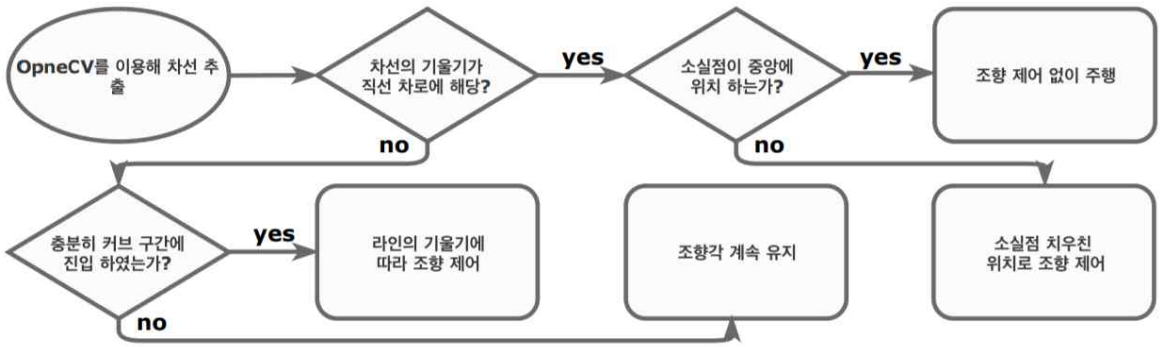
9) **OpenCV_hough_transform**
영상 이미지를 통해 차선의 hough line들을 추출하여 관련 데이터를 출력하는 역할을 한다.

10) **OpenCV_signal_detection**
Traffic Light state에서 영상으로 신호등을 인식하는 역할을 한다.

11) **OpenCV_color_detection_stopsign**
Normal Driving state에서 일시 정지 표지판이 나오면 정지할 수 있도록 영상에서 빨간색 표지판을 추출하는 역할을 한다.

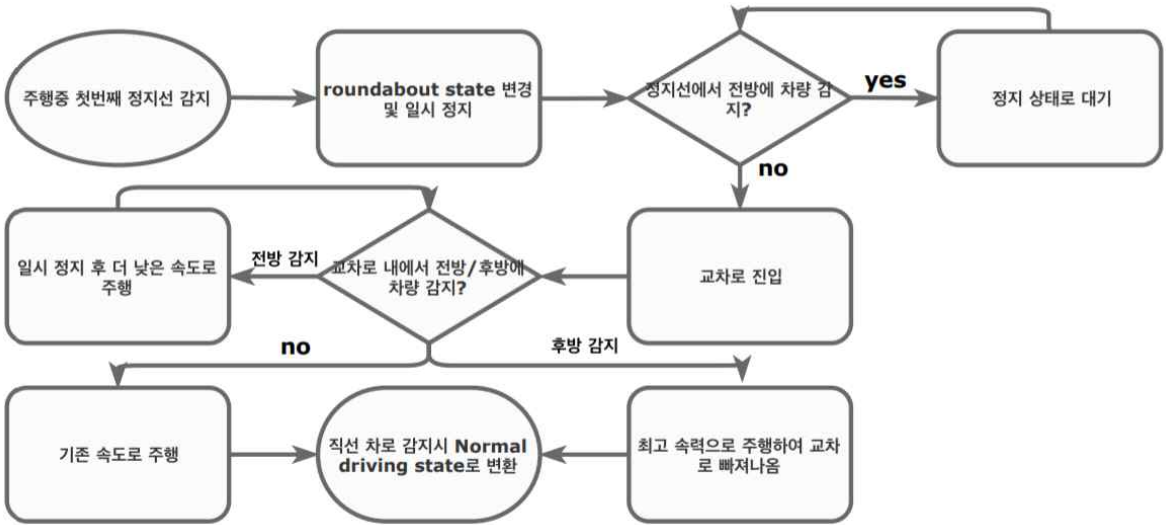
3.2. 주요 함수의 흐름도

3.2.1. state_NormalDriving() (LaneKeepingAssist( ) 함수)



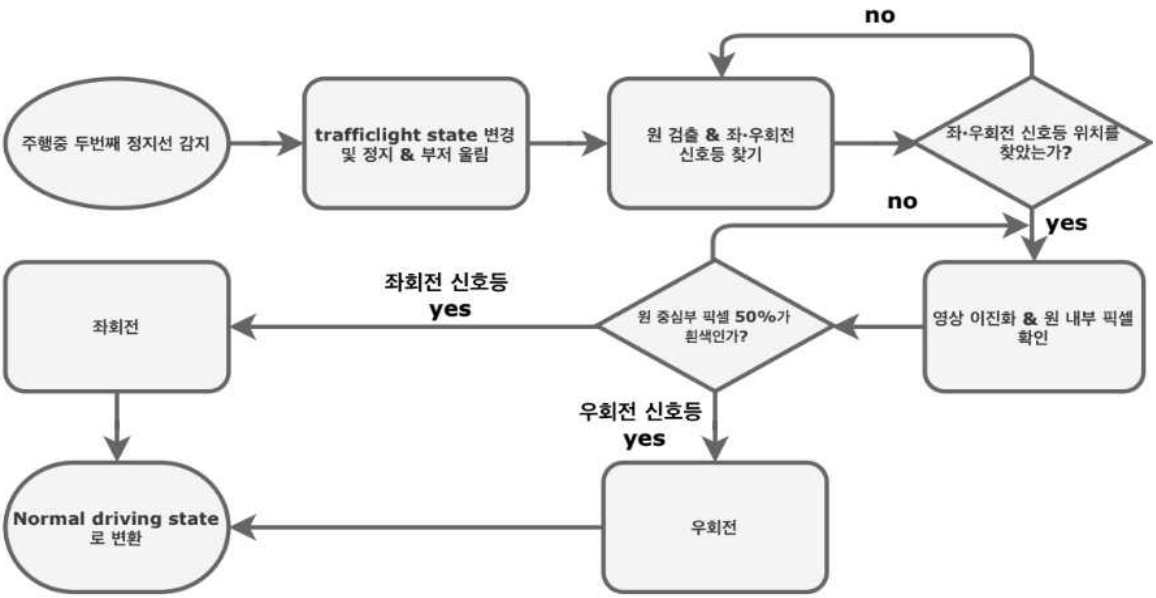
'state_NormalDriving()' flowchart

3.2.2. state_Roundabout()



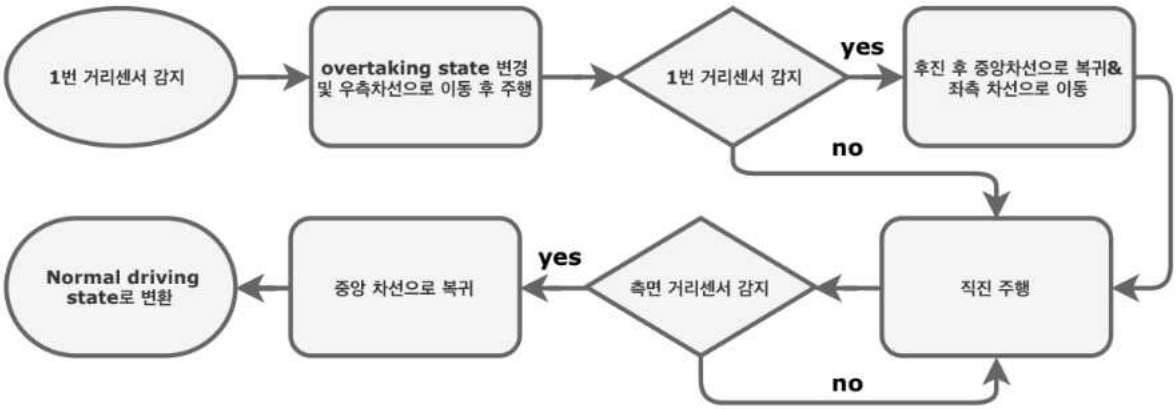
'state_Roundabout()' flowchart

3.2.3. **state_Trafficlight()**



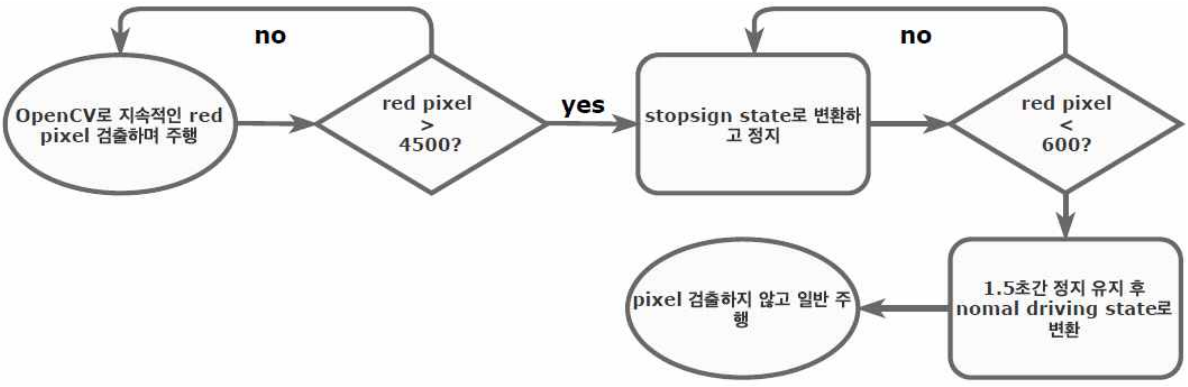
'state_trafficlight()' flowchart

3.2.4. **state_Overtaking()**



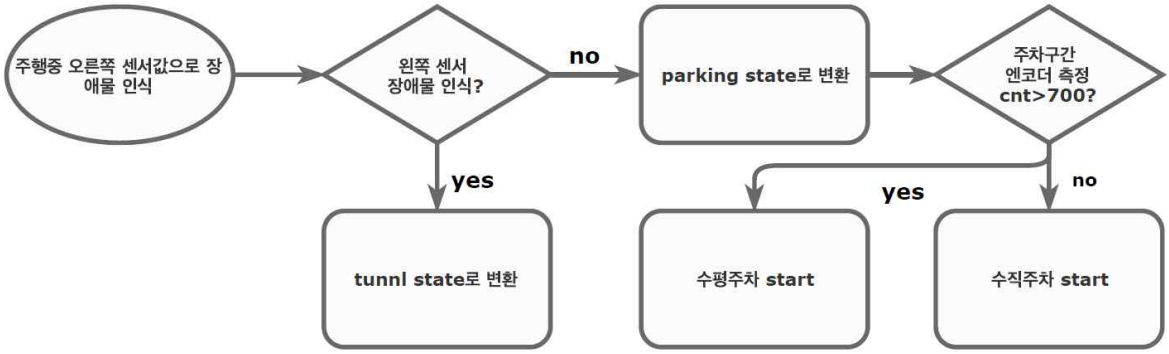
'state_Overtaking()' flowchart

3.2.5. **state_StopSign()**



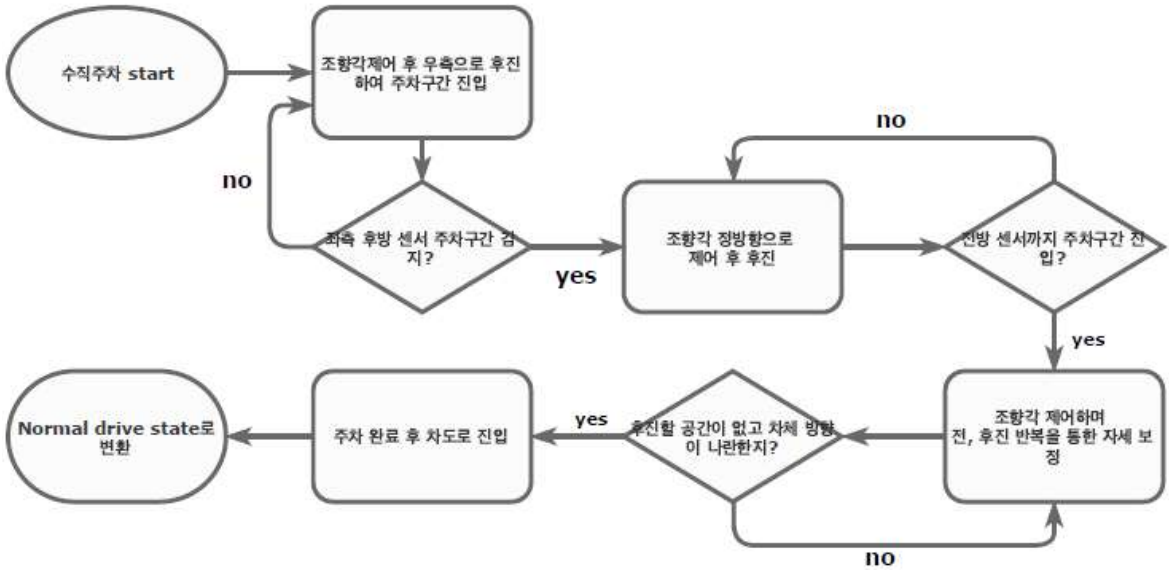
'state_StopSign()' flowchart

3.2.6. **CheckModeBySensor() (주차/터널 진입 판단 함수)**



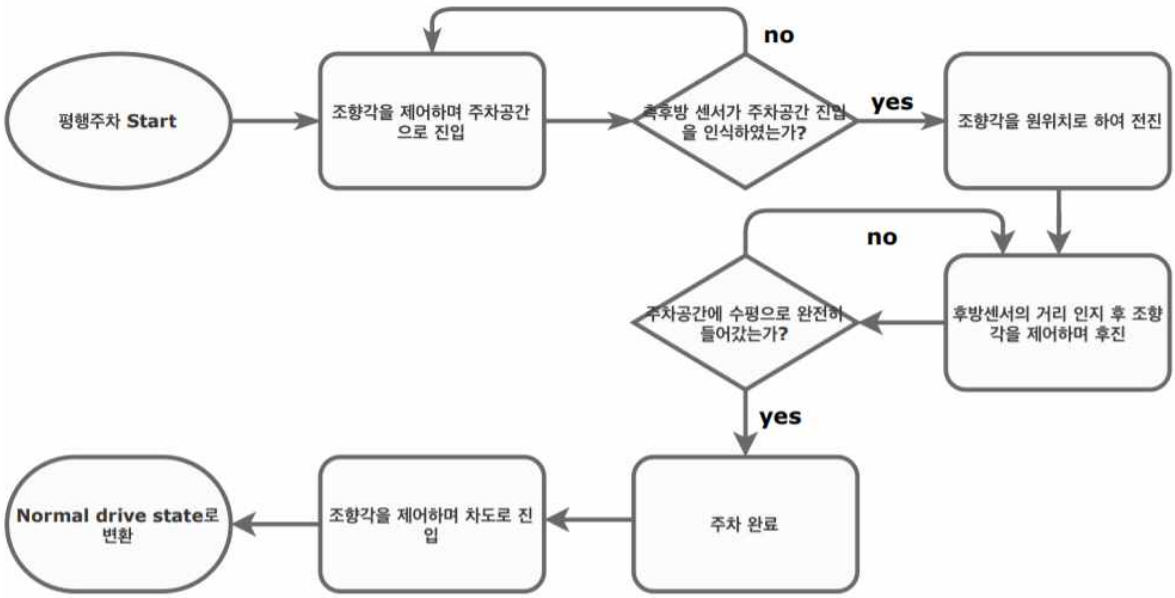
'CheckModeBySensor()' flowchart

3.2.7. **perpendicularParking()**



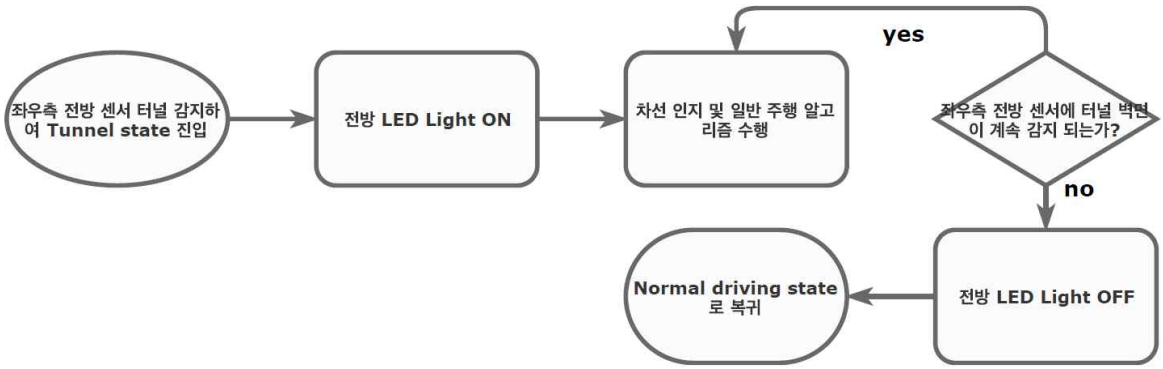
'perpendicularParking()' flowchart

3.2.8. **parallelParking()**



'parallelParking()' flowchart

3.2.9. **Tunnel_state()**



'Tunnel_state()' flowchart

3.3. 기술적 차별성

3.3.1. **차체 자세 및 치우침 보정 성능 우수**

직선 코스에서 차체의 자세가 틀어졌을 때 평행한 자세로 보정하고, 차선 한쪽으로 치우쳐져 있는 경우 차선 중앙으로 들어오게 하였다. 그로인해 다른 미션 코스에 불안정하게 진입할 수 있는 여지를 최소화 하였다.

3.3.2. **신호등 인식 효율 및 정확도 향상**

1) 매 프레임마다 신호등 원 검출->신호등 on/off 확인을 반복하지 않고, 신호등의 위치가 확정이 되면, 다음 프레임부터는 신호등 on/off만 확인함으로써 불필요한 계산을 줄였다.
2) Threshold 변환으로 신호등 on/off를 판단하여 낮은 조도환경에서도 인식률을 현저하게 높였다.

3.3.3. **일시정지 표지판의 색 추출 정확도 향상**

HSV에서 조도 변화에 밀접한 S(채도), V(명도) 값을 최소값 0에서 최댓값 255로 지정하여 색 추출을 했다. 그로인해 환경 변화에 따라 영향 받는 정도가 현저히 줄었다.

3.3.4. **주차구간에서 차체 자세 조정 정확도 향상**

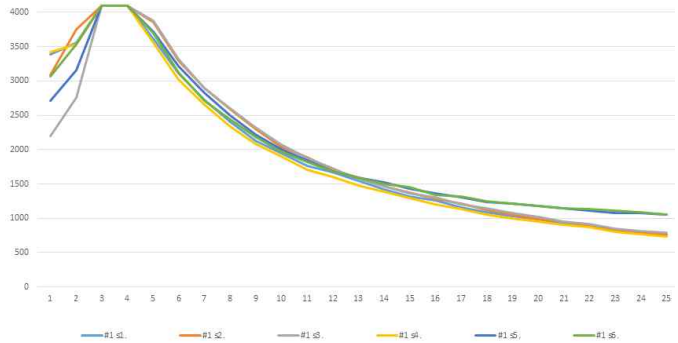
주차 구간과 차체 자세가 나란하도록 제어하기 위해 측면 전·후방 센서를 비교하여 후진, 전진을 반복한다. 정확도 높은 주차를 구현하기 위해 실제 주차와 같은 알고리즘을 구현하였다. 이로 인해 주차 구간과 나란하도록 차체 자세 정확도를 높였으며 주차 구간과 충돌 위험이 현저히 줄었다.

3.3.5. **주차구간 진입 시 충돌 위험성 현저히 저하**

주차 구조물을 인식하여 Parking State로 변환된 이후에도 일반 주행(라인 인식 영상처리) 방식을 추가 적용하였다. 영상처리 제어는 주차 구간으로 진입하기 위해 후진하기 직전까지 적용된다. 진입 직전까지 자세의 조정을 진행하여 모형차가 주차 구간에 진입하는 시점의 위치 오차를 최소화하여 안정도를 높였다.

3.3.6. **각 거리 센서 채널의 실제 거리에 대응하는 센서 값 데이터를 수집하여 제어 정밀성 향상**

IR 거리 센서 6개의 거리에 따른 데이터를 수집하여 주차 장애물에서의 조향 알고리즘을 세부적으로 제어할 수 있도록 하였다.



IR Sensor - 6개 채널의 센서 각각에 대해 수집된 데이터 기록

① 4cm~25cm사이의 거리를 10회 정도 반복하여 IR 거리 센서의 데이터를 수집하였다.

② 수집된 데이터의 평균값을 이용하여 시각화하였다.

③ 각 센서의 데이터에 따른 실제 거리를 정확히 인지하여 모형차 제어 정밀성을 높였다.

4. 개발 중 장애요인과 해결방안

'구현 시도' 항목은 문제 해결 혹은 성능 향상을 위해서 시도해 보았던 알고리즘이나, 더욱 치명적인 문제점에 직면하여 결과적으로 적용되지 못한 항목이다. 하지만 최종 적용된 알고리즘을 고안하는데 중요한 역할을 한 것들이다.

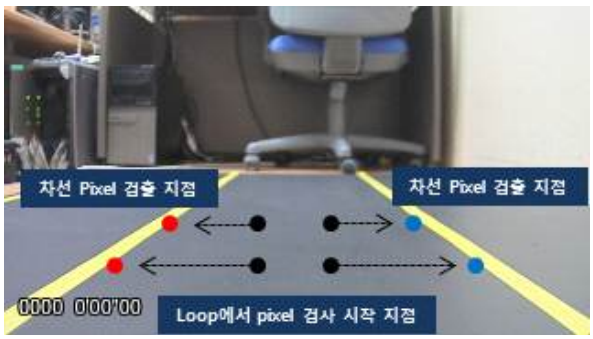
'장애 발생' 항목은 제약 조건이나 소프트웨어 혹은 하드웨어적인 한계로 인해 부딪힌 문제들이다. 주어진 미션들을 달성하기 위해 적용된 해결책을 기술하였다.

4.1. 일반 도로 주행 구현

4.1.1. 구현 시도

Pixel 단위 접근을 통한 차선 검출 및 기울기, 소실점 검출 실패

1) 상세 내용



pixel 접근 방식으로 차선 검출

현재 주행 중인 도로의 차선 바깥의 다른 차선이나 물체에 방해 받지 않기 위해 Threshold가 적용된 이미지에서 도로 중심부(픽셀 값 : 0)로부터 차선이 검출되는 지점(픽셀 값 : 255)을 찾아 차선의 기울기 및 중심점을 계산하려는 시도를 하였다.

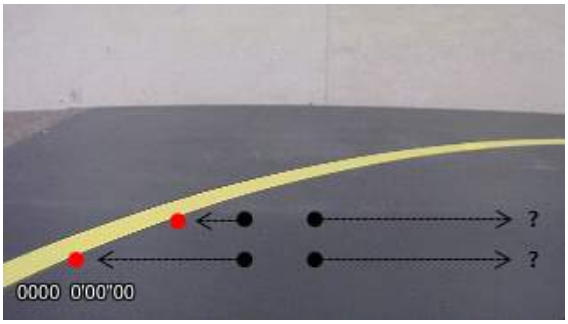
2) 기대 효과

S자 코스에서 오인식 가능 외부 차선

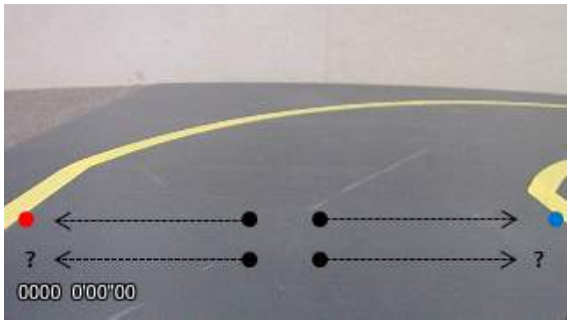
① OpenCV로 전체 이미지에 대한 변환 연산하기보다 일부 픽셀 값만을 확인하여 차선을 찾아내는 것이 연산속도가 빠를 것이라고 기대

② Hough line을 검출할 시에 S자 코스에서 발견될 수 있는 외부 차선 인식을 방지할 수 있을 것이라고 예상

3) 문제 원인



한쪽 라인이 검출되지 않을 때 픽셀 접근 동작이 많아져 Latency가 길어지는 경우



설정된 y축 기준 지점이 코너 진입 구간에서 적합하지 않을 때

① 최적화된 라이브러리인 OpenCV 함수보다 수동적으로 여러 개의 픽셀 값을 확인하는 것이 속도 면에서 더 우수하지 않은 것을 확인

② for loop를 이용해 순차적으로 Pixel에 접근하는 동작을 하다 보니 양쪽 차선 중 하나라도 검출되지 않는 상황이 발생하면 이미지 규격의 끝 부분과 가까운 곳 까지 접근할 동안 loop가 계속 돌아가게 되므로, 그 순간 영상 streaming latency가 극심하게 발생하는 것이 확인 됨

③ 트랙위에 강한 빛이 들어오게 되면 오히려 그로 인한 노이즈를 차선으로 오인식하는 경우 발생

④ y축 기준 지점을 정해 pixel 검사로 x축 지점을 검출하게 되는데 코너 구간과 직선 구간에서 모두 충족하는 y축 기준 지점을 찾기가 힘듦

4) 해결 방안

① Hough line을 추출하는 방식 사용

② ROI 영역을 더욱 최적화 시키고 카메라 각도를 더욱 아래로 기울여 도로영역 외의 것들은 최대한 감지되지 않도록 조정

4.1.2. **구현 시도**

hough line으로 차선 검출 시 평균이 아닌 기울기 median(중앙값)으로 대표 라인 검출

1) 상세 내용

차선 이미지에서 검출되는 여러 개의 hough line 중 대표 라인을 찾기 위해 모든 라인들의 rho, theta의 평균을 내어 대표라인을 검출 하던 기존 방식과 달리 라인들을 기울기 값들을 sorting 하여 그것의 중간 기울기 값을 가진 라인을 대표 라인으로 간주한다.

2) 기대 효과

hough line들의 rho, theta 평균을 구하는 방식의 최대 risk는 hough line들 중 과도하게 정상 범주를 벗어나 검출된 line이 있다면 대표 라인도 그 것의 영향을 받는 다는 것이다. 대신 median 값에 해당하는 line을 대표로 간주한다면 그러한 노이즈에 영향을 받지 않을 수 있다.

3) 문제 원인

평균값을 구해서 새로운 대표 라인을 생성하는 방식과는 달리, median 방식은 선들의 slope를 기준으로 sorting하여 그 중간 값을 가진 기존 라인을 선택하는 방식이므로 slope값과 hough line 의 양 끝 지점의 정보가 같이 묶여서 sorting 되어야 한다. 따라서 각 라인들의 data를 한데 묶어주기 위해 vector STL을 사용할 수밖에 없었다.

문제는 vector를 사용하다보니 pop, push를 자주 사용할 수밖에 없었고 메모리 관리에 치명적인 영향을 주었다. 결과적으로 segmentation fault가 발생하여 프로세스가 죽어버리는 현상 때문에 median 방식을 더는 사용할 수 없었다.

4) 해결 방안
다시 rho, theta의 평균값으로 대표 라인을 계산하는 방식을 사용하였고, 대신에 카메라 각도를 최대한 낮춰 외부 라인에 대한 영향을 최대한 줄이도록 조치하였다.

4.1.3. 장애 발생 : 좌우 커브 주행 각도 불균형

바퀴가 축에 고정되지 못한 모습

바퀴가 축에 완전히 고정된 모습

1) 상세 내용
S자 코스에서 테스트 할 때 우회전은 안정적으로 수행하지만, 좌회전 시 차체가 바깥으로 밀리면서 결국엔 코스 이탈하는 현상 발생. 소프트웨어 디버깅을 시도했으나 하드웨어적으로 불가능함을 인지하였다.

2) 문제 원인
소프트웨어적으로 좌회전, 우회전 각각 최대 조향각을 입력해주고 구동시켜 봤을 때 우회전에서는 꽤 작은 곡률을 만들어 내며 주행을 할 수 있었지만 좌회전 시에만 여전이 코스를 이탈하는 현상을 확인했다. 이 테스트를 통해, 소프트웨어의 문제거나 트랙의 곡률반경 설계의 문제가 아니라는 것을 밝혀냈다.

하드웨어 검증 결과, 모터의 힘을 받는 양쪽 두 바퀴가 기어 축에 고정되어 있지 않은 문제였다. 좌측 뒷바퀴만 모터의 힘을 받아 돌아가고 우측 뒷바퀴는 모터의 힘을 전혀 받지 못하고 헛도는 문제가 있었다. 좌회전에서 우측 바퀴는 전혀 힘을 주지 못하고 가만히 있는 상황이 되다보니 제대로 된 코너링을 하지 못하고 바깥으로 차선 이탈하게 되는 문제가 발생하였다.

3) 해결 방안
미니로봇을 통해 바퀴 교체 및 수리하여 장애 문제를 해결하였다.

4.2. 신호등 구간 구현

4.2.1. 구현시도
HSV 색 공간을 이용한 신호등 색 판별 실패

1) 상세 내용
직관적인 색 판별을 위해 RGB 영상을 HSV 색 공간으로 변환하고, 신호등 녹색 불에 대응되는 mask를 설정하여 좌우 조향 조건을 판별하려는 시도를 하였다.

2) 문제 원인
주행 공간의 조도에 따라 카메라가 인식하는 신호등 불의 색이 달라지므로, 공간에 따라 mask를 새로 설정 해주어야하는 문제점이 발생하였다.

3) 해결 방안
신호등 불이 들어오지 않았을 때는 검은색 배경이고, 이것은 녹색 불에 비해 상대적으로 어두운 색을 띤다는 점에 착안하였다. 색을 판별하는 방법 대신, 영상을 이진화 하여 일정한 threshold 값을 기준으로 신호등 불의 on/off 상태를 판단했다.

4.2.2. 구현시도
녹색 불의 픽셀 수에 따른 좌우 조향 결정 실패

1) 상세 내용
좌회전일 경우 화살표, 우회전일 경우 꽉 찬 원형의 녹색불이 켜진다. 따라서 녹색 픽셀 수를 세어 일정한 기준보다 적으면 좌회전, 많으면 우회전으로 판단하려는 시도를 하였다.

2) 문제 원인

모형 차와 신호등과의 거리가 달라짐에 따라 녹색 픽셀 수의 기준점을 다르게 잡아야 했다. (거리가 가까울 경우 좌회전 신호에서도 녹색 픽셀이 많이 검출되므로, 우회전이라고 잘 못 판단하는 경우가 생겼고, 그 반대일 경우 우회전 신호에서 좌회전 신호라고 판단하는 경우가 생겼다.)

3) 해결 방안

영상에서 무작정 녹색 픽셀 수를 세는 방법이 아닌, hough 변환을 이용해 신호등 원을 검출하고, 좌·우회전 신호가 들어오는 원의 위치를 찾아 그 원 안의 픽셀만을 검사하여 좌우 조향을 하도록 하였다.

4.3. 주차 구현

4.3.1. **장애 발생**

주차 모드진입 시 자세 틀어짐으로 인한 오류

1) 상세 내용

주차 구조물을 인식하고 Parking state로 들어간다. Parking state에 진입한 직후에는 고정된 조향값에 속도 제어만 이루어지므로 초기 자세가 틀어짐에 따라 계획한 경로에 오차가 생기는 문제가 발생한다.

2) 문제 원인

차선 인식하는 일반주행 이후 구조물이 등장하면 주차모드로 진입하고 영상인식 제어 없이 오로지 센서로만 제어를 한다. 즉, 진입 직전의 차체방향에만 의존하여 주차 구조물을 지나가기 때문에 모형차의 초기 자세가 조금만 틀어져도 전진하면서 자세 오류가 더욱 가중된다.

3) 해결 방안

일반 주행 방식과 같이 Parking state 진입 초기의 전진 과정에서 Lane Keeping Assistant(라인 인식 영상처리)를 지속적으로 적용시켰다. 영상 인식을 통해 추출된 차선에 따라 자세 보정을 하여 진입 경로 오차를 최대한 줄였다. 그 결과 충돌 위험 없이 주차 공간 안으로 안전하게 진입함을 확인했다.

4.3.2. **장애 발생**

센서 비교를 이용한 조향제어 장애발생

1) 상세 내용

측면 전●후방 센서를 비교하여 모형차를 주차구간에 나란하게 주차 하도록 구현했다. 하지만 센서 비교문의 결과 기대한 조향값과는 다르게 반대로 입력 됐다. 예를 들면, 우측 전●후방센서를 비교하면 좌회전하며 전진해야 할 모형차가 우회전하여 주차 구조물과 충돌이 일어났다

2) 문제 원인

① 측정하고자 하는 센서 위치가 4cm 이하일 때 오류가 발생했다. 장착된 센서의 스펙 상 4cm 이하의 값은 신뢰할 수 없기 때문이다.

② 측면 전●후방 센서의 차이에 따라 조향제어를 한다. 하지만 같은 거리일 때 반환하는 센서 값이 동일하지 않기 때문에 평행하지만 센서 값의 오차범위를 고려하지 못하고 잘못된 판단을 하여 조향각 제어를 하면서 충돌이 발생된 것임을 확인했다.

3) 해결 방안

우선, 신뢰성 있는 측면 센서를 판단하기 위해 좌우측 전방 센서를 비교한다. 비교 결과 선택된 측면 센서 값을 오차범위를 고려한 차이 값으로 조향제어를 하였다. 즉, 주차구간에 모형차의 자세 틀어짐 정도가 크지 않은 이상 조향각을 원위치 하여 주차를 완료하는 융통성 있는 제어방식으로 해결하였다.

4.3.3. **장애 발생**

엔코더 카운트 값이 '65278'으로 튀는 현상

1) 상세 내용

엔코더 카운터로 수직, 평행 주차 구조물 구간 거리를 측정한다. 측정된 엔코더 값으로 수직, 평행 주차 모드로 진입하도록 알고리즘을 구현했다. 하지만 수직주차 모드로 들어가야 할 때

수평주차로 진입했다. 디버깅을 한 결과 엔코더 카운터가 '65278'의 특정 값으로 지속적으로 튀는 현상이 나타났다. 수직, 평행 주차모드 구별 알고리즘에는 800이상일 경우 수평 주차로 진입하게 되어있기 때문에 65278의 큰 값으로 인해 구현한 알고리즘이 정상 동작되지 않았다.

2) 문제 원인

소프트웨어적인 문제로 의심하여 코드 디버깅 작업을 하면서 원인 분석한 결과 라인 센서와 엔코더 카운터를 동시에 읽을 때 심하게 '65278' 값으로 튀는 것을 확인하였다. 실험적으로 오류가 발생한 부분은 밝혀냈지만 '65278' 값이 튀는 근본적인 원인은 불확실했다. 하지만 추후 하드웨어 문제인 것으로 밝혀졌다.

3) 해결 방안

소프트웨어적인 문제로 판단하여 아래와 같은 조치를 취했으나 추후 HW 문제인 것으로 밝혀지고 미니로봇을 통해 수리하여 해결할 예정이다.

① 65278 발생 빈도가 낮을 경우 예외처리를 하여 엔코더 카운터 값을 다시 읽어 들이도록 했다.

② 65278 발생 빈도가 낮지만 불규칙하여 불안한 요소를 줄이기 위해 코드의 단순화하는 방향으로 재 구현했다. 포지션 컨트롤을 사용하는 부분을 필수적인 아래 2가지 사항을 제외하고 최대한 제거했다.

②-1 수직, 수평 주차 구간을 측정할 때 사용

②-2 좁은 수직 주차 구간 빠져나올 때 사용

※ 전방 센서를 인식하고 조향제어를 할 경우 수직 주차 구간은 좁기 때문에 우회전하면서 주차 구간과 충돌 위험이 있다. 반면에 후방 센서를 인식 직후 조향제어를 할 경우 최대 조향 값을 주어 우회전을 해도 라인을 벗어난다. 따라서 센서 전•후방 사이에서 조향제어를 하기 위해 포지션 컨트롤을 사용하는 것이 필수적이다.

③ 라인 센서를 읽을 때 엔코더 장애의 빈도가 높아짐을 확인했기 때문에 오류 원인 발생 부분을 분리시키도록 재구현하였다. 주차구간 내에서는 라인 센서를 읽을 필요가 없기 때문에 주차 모드로 진입 시 라인 센서를 인식 부분을 중단시키도록 예외처리를 하였다.

5. 개발결과물의 차별성

5.1. 소프트웨어 최적화

5.1.1. State machine 구조 기반으로 설계된 소프트웨어 구조

각 기능 및 코스 별로 코드를 모듈화 하여 효율적인 분업 및 디버깅이 가능하다. 상황 인지를 통한 state 이동은 'sensor_thread'라는 스레드를 이용하여 state 간의 interface 관리를 따로 담당하도록 하여 코드 내 충돌 가능성을 최소화 하였다.

5.1.2. 서로 다른 코스 간의 진입에서 발생할 수 있는 에러가 최소화 된 알고리즘

일반 주행에서 직선 및 곡선 코스에서 항상 차선과의 간격 유지하도록 설계되었다. 그러므로 트랙 위의 차량 자세와 위치가 틀어진 상태로 다른 코스로 진입할 시 발생할 수 있는 에러 발생 확률을 줄여 주었다. 따라서 커브 구간 직후 일시 정지선이나 주차 구간 등을 만나게 되어도 일정한 자세로 다음 코스로 진입할 수 있게 된다.

5.1.3. 우수한 인식 신뢰성을 갖춘 영상 인식 알고리즘

신호등 코스에서는 ROI와 구조물 패턴 인식 알고리즘을 통해 신호등 색 구분의 정확도를 극대화 하였다. 또한 빛의 조도에 영향을 심하게 받는 색 판별 대신 밝기 변화 기준으로 각 신호등의 ON/OFF를 구별하여 극 저조도 환경에서도 신호를 구분할 수 있다.

차선 인식은 터널 내 혹은 외부에서 모두 정확히 노란색 차선을 추출할 수 있도록 HSV Filter 값을 설정하였다. 여러 상황에서의 이미지를 이용해 차선 추출 테스트를 거쳐 신뢰성이 검증 됐다. 노란색 이 외의 흰색 일시 정지선 등의 물체는 철저히 배제되어 차량의 오동작 가능성을 줄였다.

5.1.4. 높은 영상 처리 속도

일반 주행 모드에서 프레임 당 처리되는 속도는 20ms 내외를 유지하며 인지되는 Data의 높은 update 속도를 유지한다. 따라서 제어 주파수를 높여도 안정적인 제어가 가능하므로 빠른 속도의 주행이 가능하다.

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

팀원	팀	역할		비고
남창호	영상팀	주행책임, 팀장	S자 주행, 회전 교차로	baseline
김보경	영상팀	미션책임, 영상책임	추월 차로, 신호등, 객체 추적	dev-feature-trafficlight
강이경	영상팀		일시정지 표시판	dev-feature-lk
강이경	주차팀	주차책임	수직 주차	dev-feature-parking-h
김윤수	주차팀		수평 주차	dev-feature-parking-v
남창호	터널팀	터널책임	광량 분석	dev-feature-tunnel
강이경	터널팀		문제 해결	dev-feature-tunnel_lk
김남우	관리팀	스크럼마스터, PM	DevOps, requests, 의사소통관리	dev, ver, master

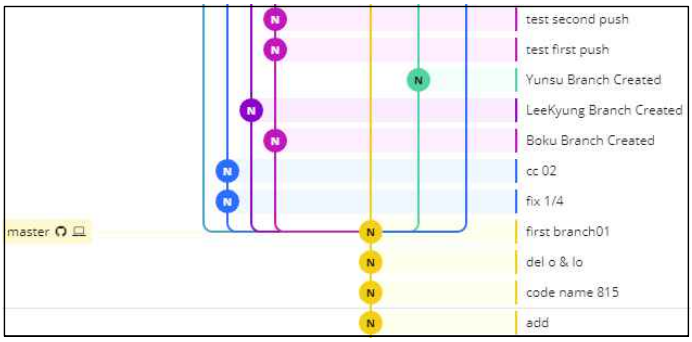
6.1. 소스 제어

* 사용 툴-Github, Gitkraken(Pro 구매)

ACE 팀 내부에 역할 별로 구성되어 다시 나눠진, 팀 속에 팀 단위로 소프트웨어를 개발하기 위해, 코드 변경 사항을 추적하고 관리를 진행해야했다. 소스 제어 관리(SCM) 시스템은 코드 개발에 대한 운영 기록을 제공하고 여러 소스의 기여 코드를 병합할 때 발생하는 충돌을 해결하는데 도움이 됐다. 코드의 수정 기록을 확인하고, 필요할 때 프로젝트의 이전 버전으로 되돌릴 수 있었고 코드를 작성하고, 준비될 때까지 자신의 작업을 격리하고, 누가 변경했고 어떤 변경 사항이 적용되었는지 파악하여 문제를 신속하게 해결할 수 있었다.

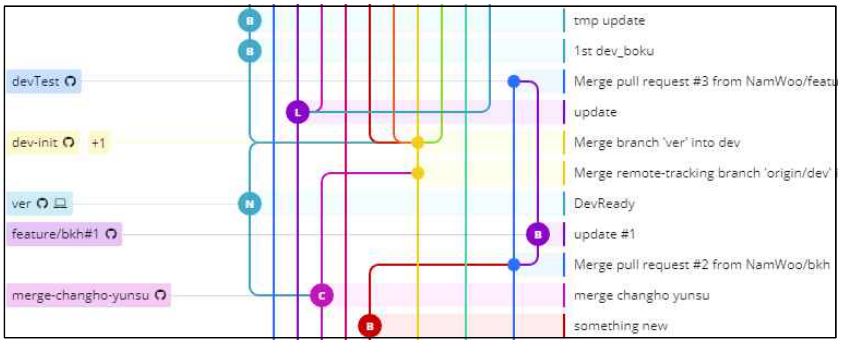
6.2. 소스 운영

6.2.1. 최초 생성



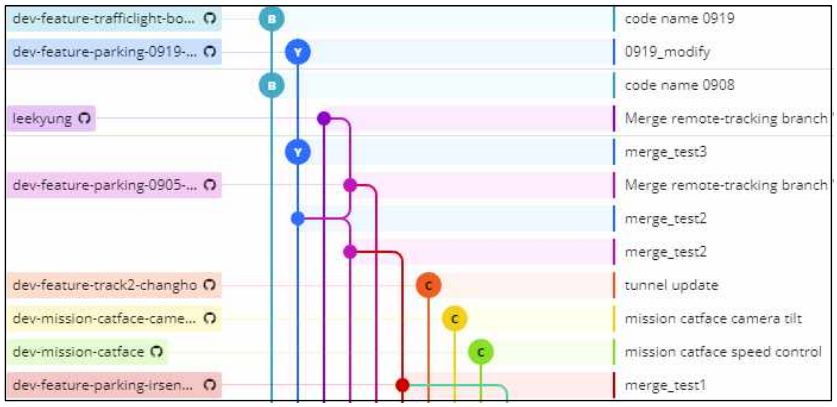
5/24 최초의 Git 생성을 시작으로 개별 학습을 진행하며 관리했고 11팀 선발 이후 각각의 Git에서 작업, 8/15 남창호 주행책임(팀장)의 베이스라인을 기준으로 소팀별로, 개인별로 branch 생성 후 개발 진행

6.2.2. 팀 단위 dev feature 배포



팀별로 진행해 오던 팀 git branch feature를 dev로 코드 병합 진행, 생기는 이슈 해결

6.2.3. dev 병합, 미션 ver



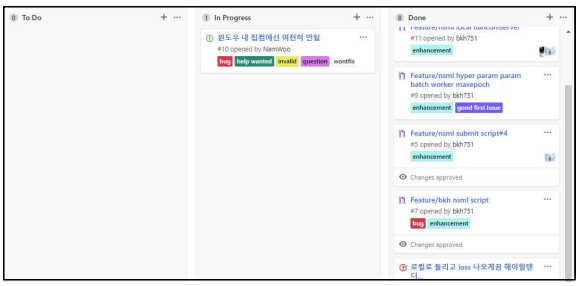
프로젝트, 미션 별로 버전 branch 관리

6.3. 프로젝트 관리

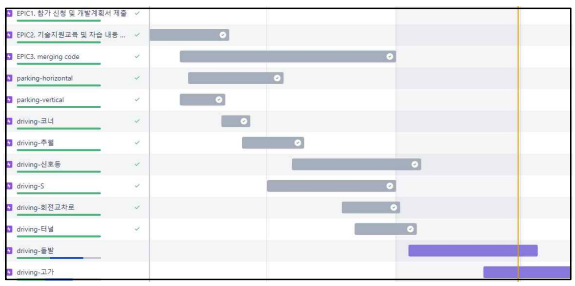
* 사용 툴–Jira Atlassian, Confluence(구매)

Atlassian JIRA + Confluence

개발 초기에는 프로젝트의 기획부터 범위, 일정, 품질, 인적 그리고 의사소통, 위험과 같은 프로젝트관리 지식체계(Project Management Body of Knowledge, PMBOK)를 기준으로 프로젝트 진행시 생기는 다양한 이슈와 방향에 대해 산출물을 만들며 진행했지만 그에 따른 기획과 준비 및 회의에 많은 시간소비가 있었고 결정적으로 하루에도 수십 번 바뀌는 코드 위주의 소프트웨어 개발 관리를 위해 애자일 방법론으로 변경, 적용하였다. 애자일은 앞을 예측하며 개발을 하기보다, 일정한 주기(일주일 기준으로 진행)를 가지고 끊임없이 스프린터를 만들어내며 그때 필요한 요구를 더하고 수정하여 하나의 커다란 소프트웨어 프로젝트를 개발해 나가는 adaptive style 이다.



Git Projects Kanban



Jira Scrum Roadmap

애자일 기법을 기획, 트래킹 할 수 있기 위해 지라(Jira) 소프트웨어를 사용했고 스크럼 방법으로 관리하며 대시보드를 활용해 팀과 개인에 따라 정보를 맞추고 조정하며 에픽을 추적했다.

EPIC 하위 이슈 관리

이슈 진행 역할분담 (담당자-보고자)

7. 결론

7.1. 팀원 간의 역할 분담 및 프로젝트 관리의 중요성

2019 임베디드 소프트웨어 경진대회 자율주행 모형자동차 부분에 5명이 한 팀이 되어 프로젝트를 진행해 보면서 체계적인 프로젝트 관리와 명확한 역할 분담을 통한 협력에 대한 역량을 키울 수 있었다. 소프트웨어 개발 방식과 방향을 팀원 전체가 협의하여 정하고, 전체 소프트웨어 구조를 설계하였다. State machine 기반의 구조로 모듈화 및 추상화한 것을 바탕으로 개발 영역을 구분하였다. 그리고 각 멤버들이 책임 분야를 명확히 설정하여 기간 별로 설정된 목표를 달성할 수 있도록 하였다. 각자 일의 진척이나 이슈사항은 Jira, Github 등을 이용하여 실시간으로 공유하며 신속히 대책을 강구할 수 있었다. 그리고 앞으로의 계획과 진척 사항을 지속적으로 상기하며 철저히 일정 관리를 할 수 있었다. 물론 처음에는 이러한 프로젝트 관리나 팀원 간의 소통이 익숙지 않았다. 하지만 매주 계획한 바를 성취해 나가며 이러한 협력 및 의사소통 방식의 중요성을 체감할 수 있었다.

7.2. 알고리즘 최적화의 중요성

범용 PC가 아닌 제한된 사양과 목적을 가진 임베디드 보드 위에서 실행되는 소프트웨어는 하드웨어가 실시간으로 알고리즘을 처리할 수 있도록 최적화하는 것이 매우 중요하다. 특히 영상 처리 알고리즘의 최적화는 카메라로 입력되는 영상의 초당 프레임 수에 결정적인 영향을 미치게 된다. 매 프레임 마다 연산 처리로 인해 스트리밍이 지연되면 차선 추출로 얻은 데이터 등이 업데이트 되는 주기가 늘어나게 되어 실시간성에 치명적인 악영향을 주게 된다. 이를 위해 이미지들을 ROI 기법으로 잘라내어 연산수를 최소화 하였고, OpenCV 라이브러리와 직접 작성한 코드를 적절히 융합하였다. 그리고 속도와 메모리 효율을 저해하는 STL 등은 전혀 사용하지 않았다. 이 과정으로부터 소프트웨어 최적화가 성능에 미치는 가치에 대해 이해할 수 있었다.

7.3. 문제 분석 및 원인 파악

개발 과정에서 소프트웨어 적인 문제뿐만 아니라 많은 하드웨어 문제도 직면하였다. 비정상적인 동작이 감지 될 때마다 충분한 검증을 통해 소프트웨어 문제인지 하드웨어 문제인지를 논리적으로 정확히 가려내야만 했다. 차량의 뒷바퀴가 축에 고정되지 못하고 헛도는 현상을 발견하기까지 코드로 조향 값을 한계치로 설정하여 여러 번의 테스트 과정을 거쳤다. 그 결과 하드웨어 자체가 커브 구간을 돌기 불가능한 상태를 인지하여 문제 원인을 파악할 수 있었다. 또한, 엔코더 카운터 값을 읽는 과정에서 65782라는 비정상적인 수가 입력될 때도 마찬가지였다. 소프트웨어 내에서 충돌할 가능성 있는 요소들을 하나씩 제거하며 테스트한 결과 아무런 충돌 요소가 없음에도 그러한 값이 출력되는 것을 확인하여 하드웨어 문제임을 밝혀냈다. 소프트웨어 혹은 하드웨어 문제 원인을 밝혀내는 과정은 앞으로 임베디드 소프트웨어 개발자로 성장하는 과정에서 필연적으로 거쳐야하는 것이다. 그래서 이 경진대회를 준비하며 겪었던 문제들을 경험할 수 있어서 감사하다.

7.4. 앞으로의 방향

임베디드 소프트웨어 경진대회 본선 진출이라는 소중한 기회를 통해 실질적으로 필요한 역량들을 확인하고 체득할 수 있었다. 필요한 기능을 구현할 수 있는 프로그래밍 역량뿐만 아니라 여러 사람들과의 의사소통, 다양한 문제를 파악하고 해결할 수 있는 능력을 키울 수 있었다. 이것을 계기로 우리나라의 임베디드 소프트웨어 및 자율주행 분야 기술 발전에 기여하는 것이 우리들의 앞으로의 목표가 되었다.