제18회 임베디드SW경진대회 개발완료보고서

[자율주행 모형자동차]

□ 개발 개요

○ 요약 설명

팀 명	영차영차
목 표	임베디드 소프트웨어 제작 기술 습득 및 실제 자율주행차에 적용 가능한 알고리즘 구현
소스코드	https://github.com/jongwook98/2020ESWContest_car_2051
시연동영상	https://youtu.be/f_BQ3XMBDCw

○ 개발 목적 및 목표

- 자율주행 모형자동차의 하드웨어를 제어하기 위한 소프트웨어를 임베디드 환경에서 제작함으로써, 임베디드 소프트웨어 제작 기술을 습득할 수 있고, 이를 통해서 자율주행의 개념을 이해할 수 있다.

- 라인인식에 대한 기능을 구현한 뒤 자율주행 상황에서 발생할 수 있는 위험요소나 그 외사용자를 편리하게 할 수 있는 여러 기능들을 논의하면서 대회에서 수행해야하는 미션들과 비교하여 실제 자동차에 적용할 수 있는 알고리즘 구현을 목표로 한다.

- 모형자동차를 다음의 세 가지 키워드를 중점으로 구현한다.

보편성 : 주어진 예제와 제공되는 OpenCV 라이브러리를 활용한 일반적인 방법으로 구현하여 진입장벽을 낮추고 활용도를 높인다.

안정성 : 카메라를 통한 영상처리와 IR 및 PSD 센서의 유기적인 상호작용을 통하여 모형자동차가 안정적인 자율주행을 할 수 있도록 한다.

유동성 : 속도 및 카메라의 각도가 직선 및 곡선의 주행상황에 맞게 유동적으로 변화하도록 한다. 또한 3차선이나 회전교차로와 같은 미션에서 장애물의 위치에 따라 회피하도록 하여 매끄러운 주행을 가능하게 한다.

□ 개발 방향 및 전략

○ 기술적 요구 사항

1. 개발 환경 구축

2. 쓰레드 구조를 통한 스케줄링

3. 소프트웨어 구성

4. 영상처리를 통한 안정적인 자율주행 구현

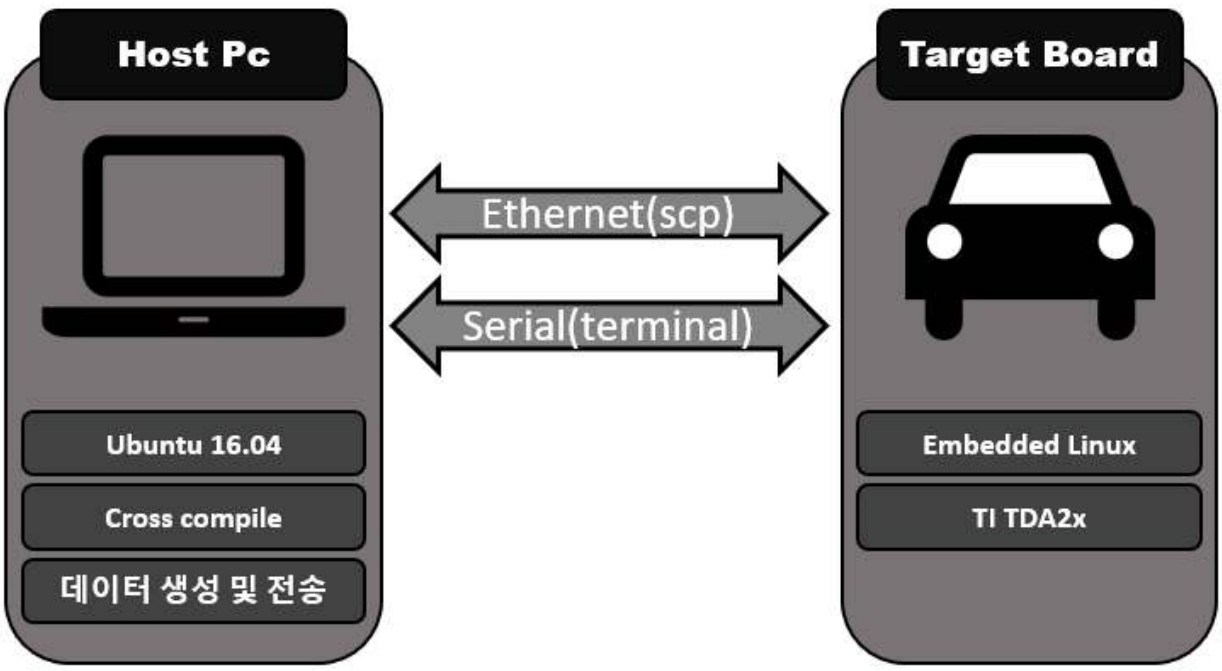
5. 각 미션별 알고리즘 구성

○ 개발 방법

1. 개발 환경 구축

1.1 개발 환경

노트북에 Ubuntu 16.04를 설치하여 Linux 환경을 제공하는 Host Pc로 활용하였고, 터미널 편집기를 통하여 소스코드를 작성하였다. 모형자동차(Target Board)는 Embedded Linux 기반의 OS(Operating System)를 사용하며, Serial 통신 및 Ethernet을 통해 Host Pc와 상호작용한다.



[그림 1-1] 개발 환경 구조도

Serial 통신을 통해 Target Board의 터미널을 확인함으로써 센서값 ,영상 픽셀값 등을 출력하여 디버깅에 활용하였으며 Ethernet을 통해 Host Pc에서 작성한 실행파일의 데이터를 Target board에 전송하였다.

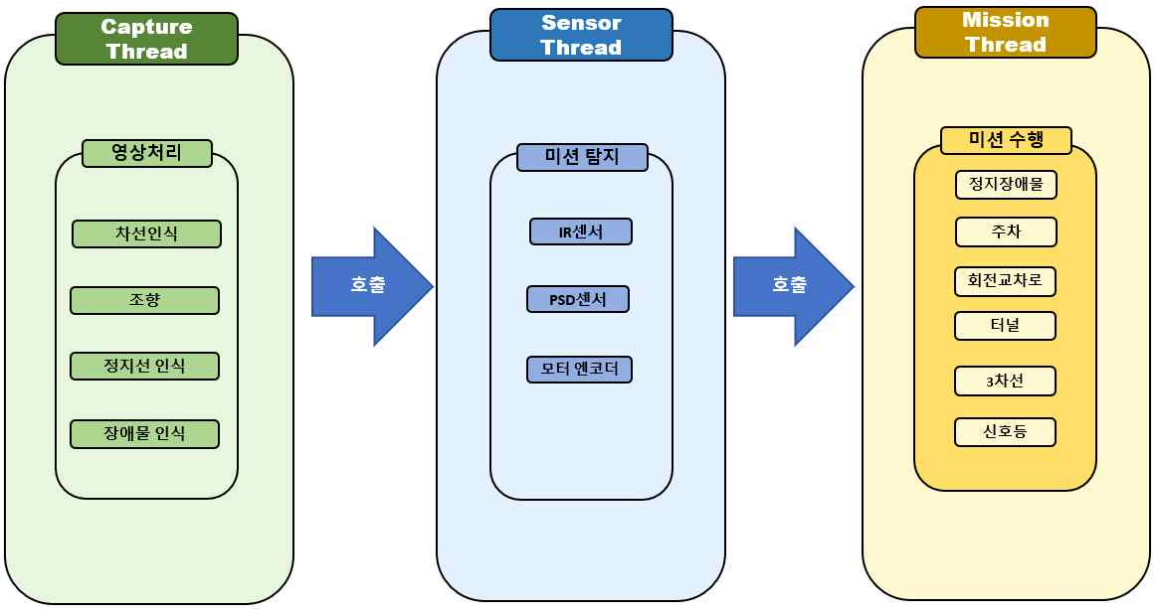
1.2 개발 언어

기본적으로 C언어를 통해 소스코드를 작성하였으며, 차선인식 및 신호등과 같이 영상처리가 필요한 부분은 C++언어를 활용하였다. 이때 주어진 OpenCV 라이브러리의 함수를 적극적으로 활용하여 작성하였다.

2. 쓰레드 구조를 통한 스케줄링

주어진 예제에서 dump thread와 input thread는 주행 시엔 필요하지 않기 때문에 제거하였고, 몇 가지 쓰레드를 추가하여 스케줄링 하였다.

쓰레드간의 메모리충돌을 막기 위해 mutex 잠금을 통해 동기화하였으며, 조건변수를 활용하여 쓰레드 간의 호출에 이용하였다. 이때 각 쓰레드는 잠금 되었다가 호출에 따라 활성화와 비활성화를 반복한다. 전체 쓰레드 구조는 capture_thread, sensor_thread 그리고 각 미션 쓰레드들로 구성된다.

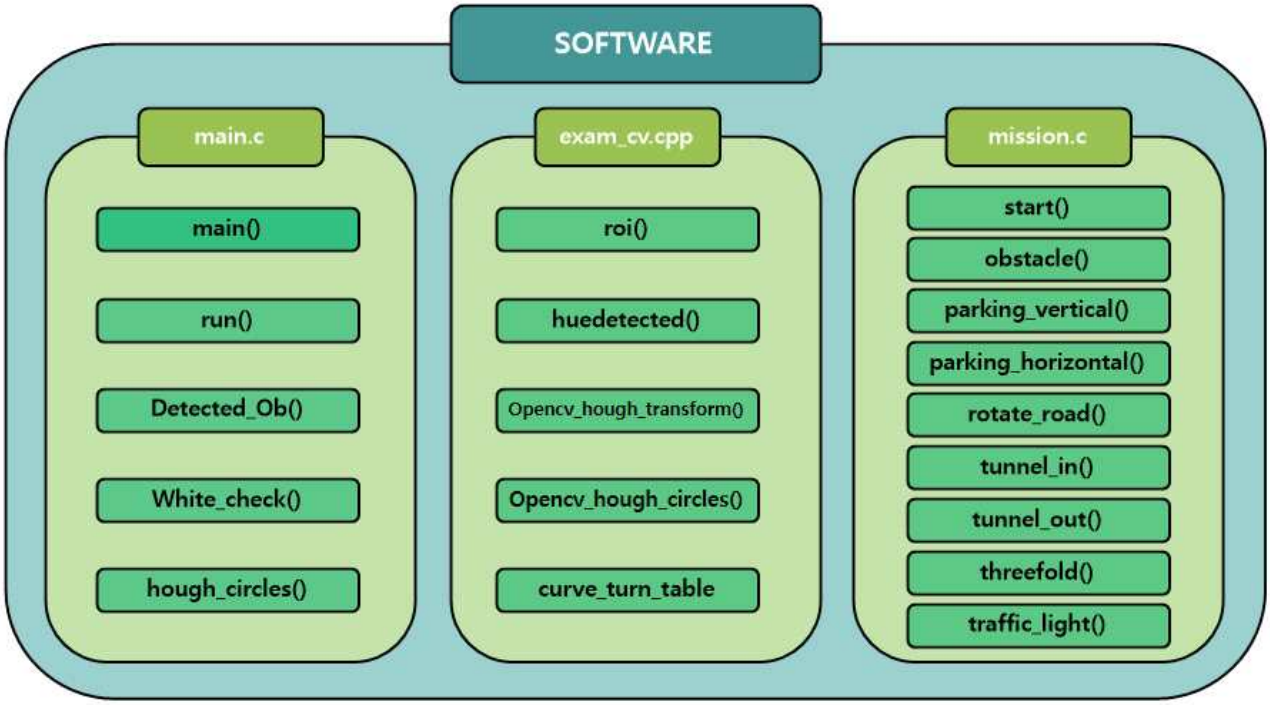


[그림 2-1] 쓰레드 구조도

capture_thread는 영상처리 및 주 쓰레드로써 일정 주기마다 sensor_thread를 호출한다. 호출에 따라 sensor_thread가 라인 감지용 IR 센서, 벽을 감지하는 적외선 PSD 센서를 통해 미션을 검출한다. 이때 mode_count에 따라 해당하는 mission에 대한 센서값만을 검출하도록 한다.

조건을 만족하면 각 미션에 해당하는 mission thread를 호출한다. 각 mission thread는 미션을 수행하고, 스스로 종료한다.

3. 소프트웨어 구성



[그림 3-1] 소프트웨어 구조도

소프트웨어는 크게 쓰레드의 호출과 주행함수를 포함하는 main.c 파일, 영상처리 관련 함수들을 포함하는 exam.cpp 파일, 미션함수를 포함하는 mission.c 파일로 구성되어 있다.

3.1 main.c

3.1.1 파일요약

주행에 필요한 함수들로 구성되어 있으며, Carcontrollinit() 및 variable_init()을 통해 주행에 필요한 기본 조건들을 초기화한다. 또한 각각의 쓰레드를 생성하며, capture thread에서 주행 함수들을 호출하여 주행을 시작한다.

3.1.2 함수설명

run() : 검출된 차선의 값을 바탕으로 조향각과 속도를 결정한다.

Detected_Ob() : 우선 정지 장애물을 탐색하고, red 픽셀 값을 얻는다.

White_check() : 흰색 정지선을 탐색한다.

hough_circles() : 신호등 미션 수행 시 원을 검출하고, 신호등의 색을 구분하여 신호등의 분기를 판단한다.

3.2 exam_cv.cpp

3.2.1 파일요약

영상처리에 관련된 함수로 구성되어 있으며, 각 함수들은 외부 전역변수를 통해 조건에 맞게 실행된다.

3.2.2 함수설명

roi() : 연산속도를 높이고 불필요한 부분을 없애기 위하여 관심영역을 설정한다. 이때 조건에 맞게 가변적으로 영역을 설정하도록 한다. 차선인식의 경우 좌, 우 차선의 관심영역을 각각 설정하여 차선을 잘못 인식하는 경우를 최소화하였다.

huedetected() : yellow, white, red 등 hsv 영역의 색을 추출한다.

OpenCV_hough_transform() : 추출한 hsv 영역에서 houghlines 알고리즘을 통하여 라인을 검출하고, 이를 바탕으로 조향 각을 결정한다.

OpenCV_hough_circles() : 특정 임계값 이상의 원을 검출하고, 신호등 분기점의 신호를 검출한다. 검출한 원 내부에서 hsv영역의 픽셀 값으로 신호등을 구분한다.

curve_turn_table : 차선 기울기에 따른 가중치를 변경하여, 곡선 주행 시 올바르게 조향하도록 한다.

3.3 mission.c

3.3.1 파일요약

미션 수행 시 호출하는 함수들로 구성되어 있으며, 각 미션이 종료되면 mode_count++을 통해 다음 미션으로 넘어가게 된다.

3.3.2 함수설명

obstacle() : 우선정지 장애물의 해당하는 red 픽셀 값을 검출하여 정지 여부를 판단한다.

parking_vertical(), parking_horizontal() : 주차공간을 확인하고 모터 엔코더 값의 조건을 통해 수직, 수평 주차임을 판단하여 조건에 해당하는 함수를 실행한다.

rotate_road() : 회전교차로 미션을 수행한다. PSD센서를 통해 전방, 후방 거리를 확인하여 회전차량과 충돌을 방지하고, 흰 선을 주행차선으로 인지하여 안정적으로 탈출한다.

tunnel_in(), tunnel_out() : 터널의 진입과 탈출을 확인한다. 좌, 우측 PSD센서를 통해 터널 진입여부를 판단한다.

threefold() : 차로 추월 구간에 진입한 뒤 전방에 장애물이 감지되었을 때 불러오는 함수이다. 장애물의 위치에 따른 추월방향을 판단하여 장애물을 추월한다.

traffic_light() : 3차로 미션의 종료 후 정지선을 감지하면 불러오는 함수이며, hough_circles 함수를 이용하여 검출된 원을 색 검출을 통해서 신호를 구분하고, IR 센서를 사용해 정차 구간을 감지하여 주행을 종료한다.

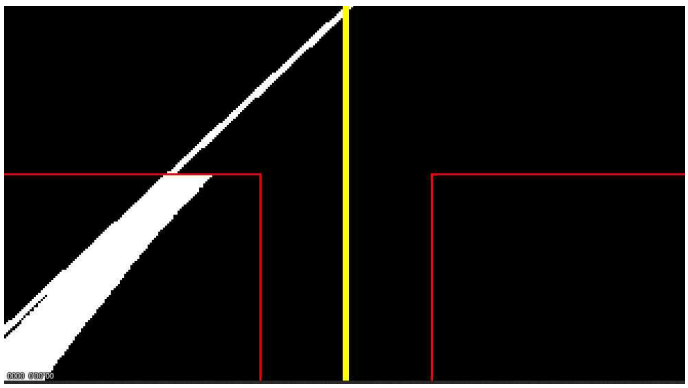
4. 영상처리를 통한 안정적인 자율주행 구현

차선을 인식하는 과정에서 OpenCV 라이브러리 함수들을 적극적으로 활용하여 구현하였다. 노란색 차선을 추출하기 위해 inRange 함수를 사용하였고, Canny 함수로 경계선을 추출하였다.

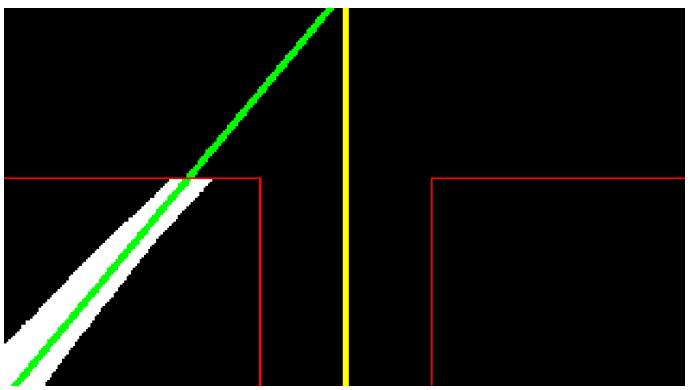
4.1 차선인식

차선 검출에 앞서 왼쪽 차선과 오른쪽 차선의 검출이 예상되는 구간에 대하여 따로 ROI(관심영역)을 설정한다. 각각의 ROI에서 노란색 영역을 검출한 뒤 HoughLines 알고리즘을 사용하여 직선을 추출하며 좌표 값에 따라 해당 ROI 영역에 맞는 Line인지 아닌지를 구분하여 값의 저장을 결정한다.

차선 검출을 잘 하기 위해 임계값을 낮추어 다량의 선을 검출하여 평균값을 도출하였다. 도출한 직선의 좌표 값에 따라 Base Line([그림 4-1] 참조)과 Edge Line([그림 4-2] 참조)으로 구분하였으며, 검출된 영역에서 차선의 기울기 값을 참고하여 잘못 검출되었을 경우에는 무시하였다.



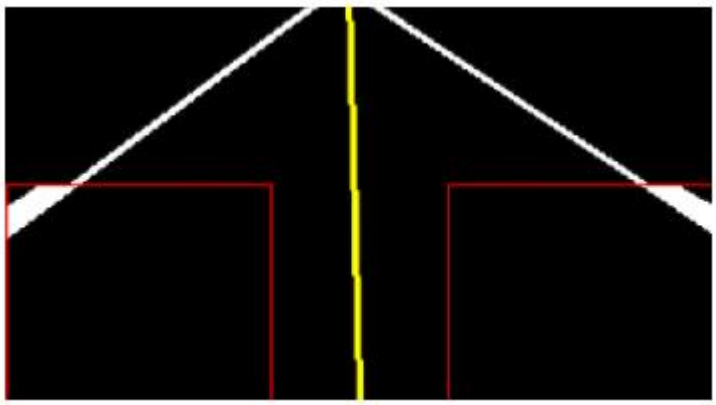
[그림 4-1] Base Line 검출 (흰색)



[그림 4-3] Edge Line 검출 (연두색)

4.2 직선주행

일반적으로 두 차선이 검출되는 경우 소실점 제어를 통하여 조향하였고, 한쪽 차선이 검출되는 경우에는 차선의 기울기, 치우침에 따라서 보정하여 중앙으로 가도록 하였다.



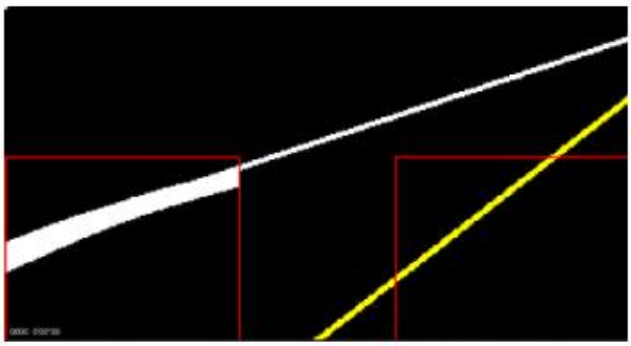
[그림 4-3] 좌우 소실점을 통한 조향각도 제어

[그림 4-3]에서 하얀색 직선은 검출된 차선을, 노란색 직선은 소실점에 대한 진행방향을 나타내며, 빨간색 영역은 각 차선의 ROI 영역이다.

직선 구간에서 한 쪽 차선만 검출되는 경우에는 차선의 좌표와 기울기 값을 이용하여 중앙으로 향하도록 하고, 차량이 라인의 한쪽으로 치우칠 경우 검출되는 Edge Line을 통해 보정한다.

4.3 곡선주행

한쪽 차선으로부터 얻어지는 치우침이 일정 기준을 넘어서면 곡선으로 간주한다. 곡선주행을 시작하면 카메라를 조정하여 곡선 주행 시 빠른 속도에도 안전하게 곡선을 탈출하게 하였다. 이 때 최대곡률에서 검출되는 직선의 y좌표를 구한 뒤 검출된 직선에서 y좌표에 해당하는 x좌표를 바탕으로 조향한다. 특히 x좌표의 값에 따라 차선의 각도를 예상하여 그에 맞는 가중치를 곱해줌으로써 안정적인 턴을 가능하게 하였다.



[그림 4-2] 곡선 인식

5. 각 미션별 알고리즘 구성

5.1. 고가도로 및 내리막

고가도로에서 주행은 기본 주행과 같으며, 좌우 PSD 센서를 이용해 고가도로의 종료를 판단한다. PID 제어를 통해 내리막 구간에서 안정적으로 주행하도록 하였으며, 특히 P제어 gain 값을 높여주어 내리막에서도 최대한 같은 속도를 유지할 수 있도록 하였다.

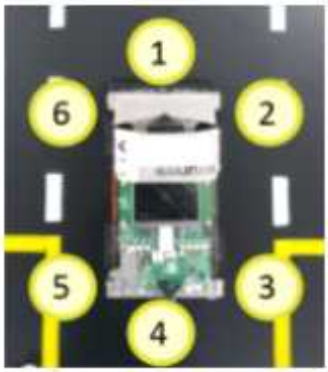
5.2 우선정지 장애물

장애물의 red 픽셀 값으로 장애물을 판단한다. 픽셀 값이 어느 정도 들어오기 시작하면 우선정지 구간이라고 판단한다. 픽셀 값이 일정 임계값을 넘어서면 정지하고, 픽셀 값이 떨어지면 장애물이 사라진 것으로 판단하여 다시 주행을 시작한다.

[그림 5-1] 우선정지 장애물 인식

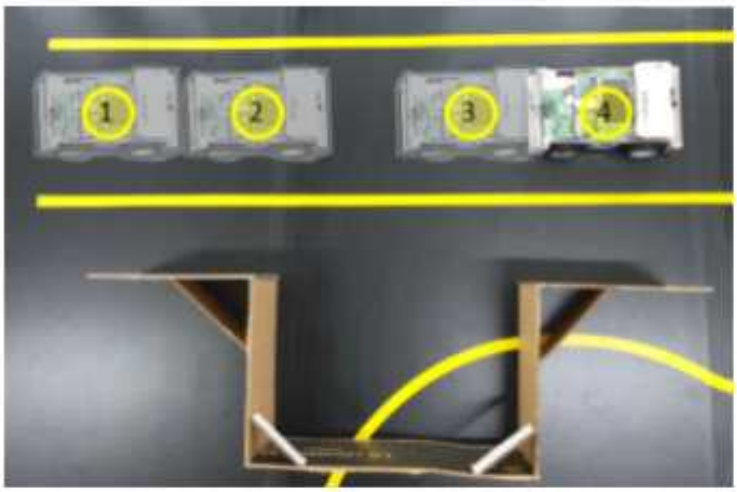
5.3. 주차

주차미션을 수행하기 위해서는 먼저 주차공간에 대한 확인이 필요하며 주차공간임을 인지하면 종류에 대한 판별도 필요하다. 이를 해결하기 위해 PSD 센서와 엔코더 값을 같이 사용하였다.



[그림 5-2] PSD 센서의 위치

5.3.1 주차 공간 판단



[그림 5-3] 주차공간 판단과정

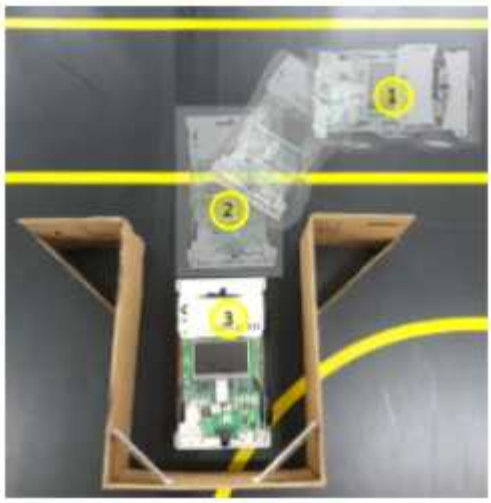
① 우측 2번 PSD센서와 우측 3번 PSD센서를 이용해 벽이 있음을 감지하면 이곳이 주차구역임을 인지시킨다.

② 우측 2번 PSD센서를 이용해 주차구역의 벽이 사라지면 엔코더 값을 초기화 하고, 누적측정을 시작한다.

③ 우측 2번 PSD센서에 다시 벽이 감지가 되면 누적된 엔코더의 값을 통해 수직주차를 실행할지, 수평주차를 실행할지 판단한다.

④ 우측 3번 PSD센서에 벽이 감지가 되면 정지 하고, 주차 미션을 수행한다.

5.3.2 수직주차 (parking_vertical)



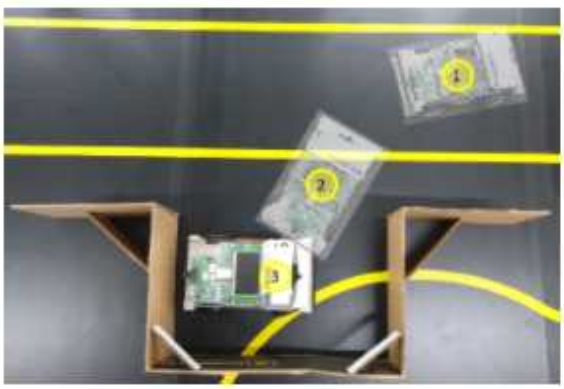
[그림 5-4] 수직주차 과정

① 바퀴를 우측으로 최대로 꺾은 뒤 후진한다. 좌측 5번 PSD 센서에 일정이상의 값이 들어오면 멈추고, 방향이 정면을 향하도록 바퀴를 돌린다.

② 후진하면서 후방 PSD 센서에 일정이상의 값이 들어오면 정지하고, 부저를 울려 주차가 완료되었음을 알린다.

③ 주차를 진행한 순서의 역순으로 다시 진행하여 주차 공간을 탈출하고, 다시 주행한다.

5.3.3 수평주차 (parking_horizontal)



[그림 5-5] 수평주차 과정

① 우측으로 어느 정도 꺾은 후 후진한다. 후방 PSD 센서가 일정이상의 값을 감지하면 멈춘다.

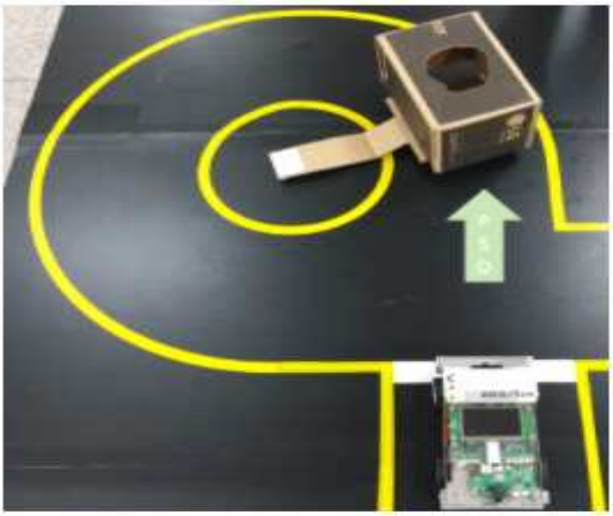
② 좌측으로 최대로 꺾은 후 후진한다. 후방 PSD 센서가 일정이상의 값을 감지하면 정지하고, 부저를 울려 주차가 완료되었음을 알린다.

③ 주차를 진행한 순서의 역순으로 다시 진행하여 주차 공간을 탈출하고, 다시 주행한다.

5.4 회전교차로

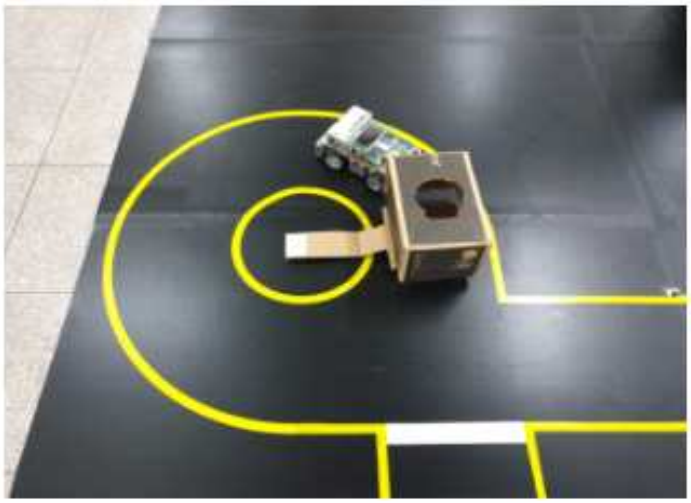
회전교차로에서 회전하는 차량에 대해 일정한 거리 값을 얻기 위해서는 항상 일정한 위치에서 정지해야 한다. 따라서 주행 도중 흰 선을 인지하면 감속하고, IR 센서를 이용하여 정지선을 검출하도록 하였다.

정지선을 인식하면 일시정지하고, rotate_road()함수를 호출하여 회전교차로 미션을 수행하기 시작한다. 이후 전방 PSD센서를 통해 회전차량을 인식하고, 회전차량이 앞을 지나가고 나서 출발한다.



[그림 5-6] 전방 회전차량 인식 후 출발

출발 후에 모터 엔코더 값을 이용하여 특정거리만큼 전진하고 일정시간동안 회전차량을 기다린다. 그 후 후방 PSD 센서를 이용하여 다가오는 회전차량이 인식되면 다시 주행한다.

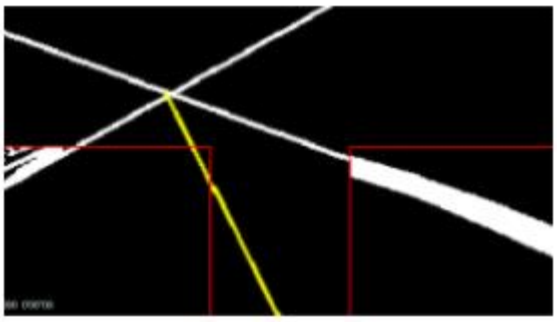


[그림 5-7] 후방 화전차량 인식 후 출발

회전교차로 미션을 수행 중일 때에는 전방 및 후방센서를 통해 회전차량의 거리를 계속 확인하여 충돌을 방지하였다. 또한 회전교차로에 진입한 순간부터 흰 선을 주행차선으로 인지하여 정상적으로 차량의 진출이 가능하게 하였으며 이후 차선의 기울기를 통해 회전교차로를 탈출했다고 판단하여 미션의 종료를 인지한다.

5.5 터널

터널에 진입하게 되면 전방 라이트를 작동시키는데, 이때 라이트의 밝기로 인해 [그림 5-8]과 같이 잡음이 발생하여 차선을 제대로 인식하지 못한다. 따라서 터널 진입 시 차선의 hsv 임계값을 변경하여 터널 안에서도 동일한 주행이 가능하도록 하였다.



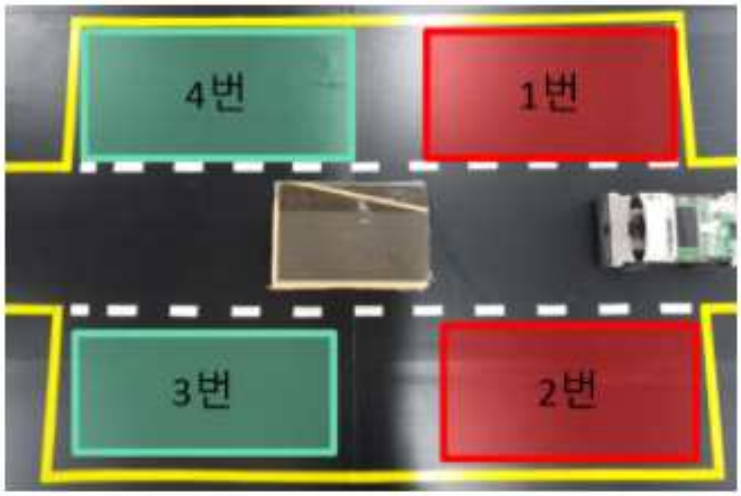
[그림 5-8] 터널 진입 시 라이트의 밝기로 인해 검출되는 잡음

터널의 진입 및 탈출 여부는 좌, 우측 PSD 센서로 판단한다. 터널을 탈출하게 되면 차선의 임계값을 기본 값으로 변경하여 다시 주행한다.

5.6 차로 추월 구간

차로 추월 구간에서 장애물 차량이 있는 위치는 유동적이므로 추월에 대한 상황이 많은 편이다. 따라서 전 방위의 PSD 센서를 이용하여 장애물의 위치를 탐색하고 추월의 방향을 판단하도록 하였다.

터널을 빠져나온 뒤 주행을 하면서 하얀색 점선이 검출된 경우 차로 추월 구간에 진입한 것으로 판단한다. 전방의 장애물을 감지하면 주행을 멈추고, 안정적인 차선변경을 위해 뒤로 일정거리 후진 한 뒤 threefold() 함수를 실행한다. PSD 센서를 통해 장애물의 위치를 확인하고 장애물이 없는 곳으로 추월한다.



[그림 5-9] 장애물 판단 구역

[그림 5-9]와 같이 장애물이 있는 예상 지점을 4가지로 나누어 판단하고, 그에 맞게 미션을 수행한다.

5.6.1 1번과 2번 구역에 장애물이 위치한 경우

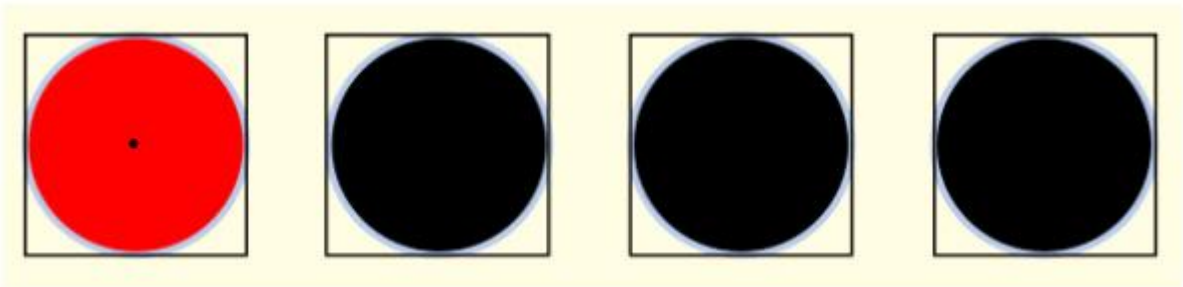
3차로 진입 후 중앙차선에서 장애물을 감지했을 때 정지한 뒤 후진하면서 장애물이 존재하는지 확인한다. [그림 5-2]를 기준으로 2번 PSD 센서에 감지가 되면 1번 구역에 장애물이 있다고 판단한 뒤 왼쪽 추월 주행, 6번 PSD 센서에 감지가 되면 2번 구역에 장애물이 있다고 판단한 뒤 오른쪽 추월 주행을 시작한다.

5.6.2 3번과 4번 구역에 장애물이 위치한 경우

5.6.1번 과정에서 장애물을 감지하지 못한 경우, 후진을 완료한 뒤에 전방 PSD 센서가 3번 구역을 감지할 수 있게 하여 장애물의 유무를 파악한다. 만약 장애물이 있다면 오른쪽 추월주행, 없다면 왼쪽 추월 주행을 진행한다.

5.7 신호등 분기점 및 도착점 정지

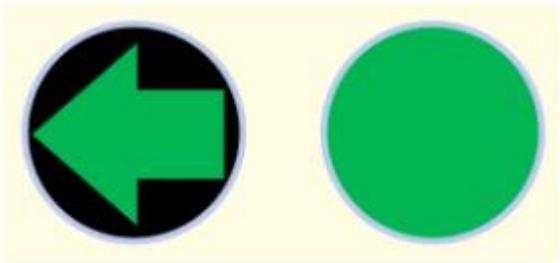
3차선 미션 종료 후 IR 센서를 통해 정지선을 감지하여 차량을 정지시키고, 카메라의 각도를 조정하여 신호등을 인지한다. 영상처리를 통한 원 검출을 통해 신호의 위치를 파악하며 각각의 중심으로 부터 원 크기의 픽셀 수를 확인하여 특정색의 픽셀 수가 기준에 미치면 해당 신호로 인지하게 한다. 신호의 움직임을 통해 일어날 수 있는 오류는 일정 시간동안 같은 신호로 판단했을 때 그 신호에 맞추어 동작하는 것으로 해결하였다.



[그림 5-10] 신호등 검출 화면 예시

위 방법으로 검출된 신호를 통해 적색 신호등을 감지하고, 청색 신호를 파악하여 이동 방향을 저장한 후 카메라 각도를 내린 후 서서히 출발한다.

청색 화살표와 청색 원신호의 구별하는 방법으로 청색의 픽셀 수를 참고하였으며 청색 신호들의 넓이를 계산해 보았을 때 청색 원형 신호가 화살표 신호의 약 두 배 크기로 검출된 픽셀 또한 대략 두 배 이상의 수가 검출되었으며 이를 이용하여 신호를 구분하였다.

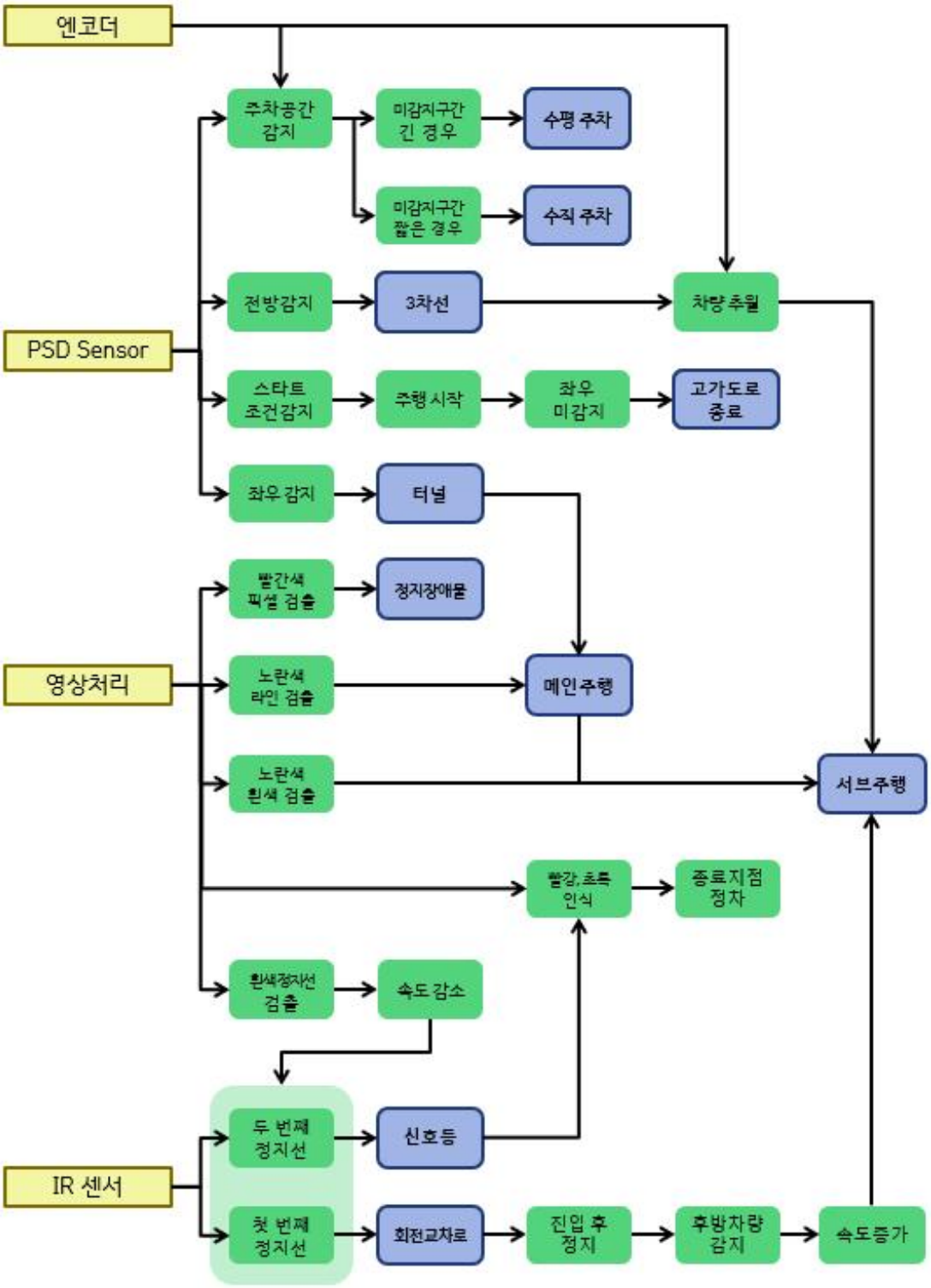


[그림 5-11] 청신호의 구분

이 후 영상처리를 통해 분기점을 인지하여 이동 방향에 따라 각도를 바꾸었으며 신호등 앞의 노란색 차선의 기울기 값을 참고하여 정차해야 하는 구간을 정면으로 바라볼 때 직진하면서 IR 센서를 통해 정지선을 감지하면 몇 초간 직진을 하고 주행을 완료하여 종료하였다.

5.8 미션 수행 흐름도

각 센서를 통한 미션 판단 및 진행을 [그림 5-13]과 같은 흐름도로 표현하였다. 미션 판단 및 진행에 실행되는 함수들은 [3. 소프트웨어 구성]에 설명되어있다. [그림 5-13]의 메인 주행은 일반적인 주행(라인 추종)을 나타내며 노란색만 검출하고, 서브 주행은 미션 판단을 통해 미션을 수행하는 주행을 나타내며 노란색과 흰색을 동시에 검출한다.



[그림 5-13] 미션 수행 흐름도

□ 개발 중 장애요인과 해결방안

장애요인	해결방안
곡선구간에 진입하는 위치가 매번 달라 곡선 주행 시 라인을 밟는 문제 발생	검출한 edge 라인을 통해 차량이 항상 차선 중앙으로 회귀하도록 하여 해결
회전교차로 진입 전 흰색 정지선이 간혹 인식되지 않는 경우 발생	영상처리를 통해 흰색 정지선이 파악된 경우에 감속한 뒤 센서값을 받도록 하여 해결
주차미션 수행을 위한 주차공간 탐색 및 판별을 하던 중 PSD센서만을 이용하여 계산했을 때 부정확해지는 경우가 발생	추가적으로 엔코더 값을 이용하여 주차공간을 측정함으로써 수직, 수평주차를 판단하도록 하여 해결
흰색 라인을 인식하는 경우 빛으로 인해 발생하는 노이즈를 차선으로 잘못 인식하는 경우 발생	ROI 영역의 조정을 통해 차선이 없을 것으로 예상되는 영역을 배제하여 해결
터널미션 진행 중 전조등을 키면 빛으로 인해 주행 라인의 임계값이 달라져 주행에 문제 발생	터널에 진입한 경우 노란색 라인의 임계값을 변경하여 터널 안에서도 라인을 정확하게 인식하도록 하여 해결

□ 개발의 차별성

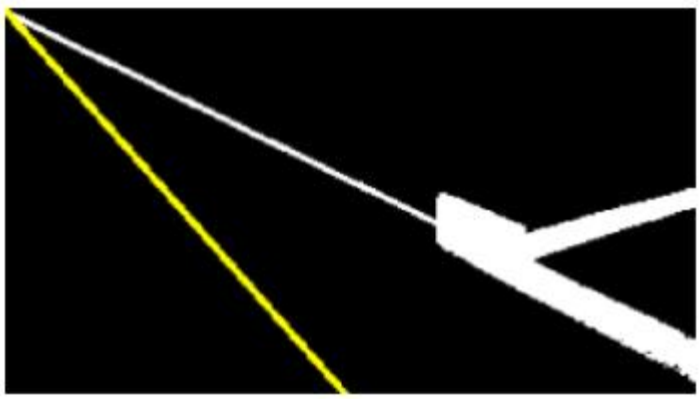
1. 효율적인 소스파일 구성

소스코드를 작성하는데 있어서 효율적이고, 시간을 단축하기 위해서는 가독성이 좋고, 값을 수정하는데 있어서도 편리해야 한다.

따라서 프로젝트 파일을 구성할 때 헤더파일들(variable.h, struct.h)을 이용하여 변수 및 구조체 선언을 따로 분리해 둠으로써 변수들을 한꺼번에 관리하도록 하였다.

2. 가변 ROI 설정

차선을 검출하는데 있어서 각 차선 구간에 대한 ROI를 달리 설정하여 차선이 교차되어있는 곳에서도 별다른 조치나 변동사항 없이 정상적으로 주행하게 하였다. 또한 정지선 인식을 위한ROI를 넓게 설정하여 정지선을 밟기 전에 안정적으로 감속하도록 하였다.



[그림 2-1] 차선이 교차되는 부분

기울기 값이 ROI 영역과 맞지 않는 라인에 대해서 무시함으로써 [그림 2-1]]과 같이 차선이 교차되는 부분은 무시하고 직선을 검출하도록 하였다.

□ 개발 일정

No	내용	6月				7月				8月				9月				10月	
1	개발환경 구축																		
2	코스제작																		
3	각종 모터, 센서 제어																		
4	차선 인식																		
5	주행 알고리즘																		
6	정지 장애물 인식																		
7	수평, 수직주차																		
8	회전교차로 주행																		
9	차로 추월 구간																		
10	신호등인식																		
11	터널주행																		
12	고가도로 주행																		
13	알고리즘 통합																		
14	디버깅 및 안정화																		

□ 팀 업무 분장

No	구분	성명	참여인원의 업무 분장
1	팀장	김진성	○ 소프트웨어 구조 설계 ○ 알고리즘 통합 ○ 정지장애물 알고리즘 구현
2	팀원	김종욱	○ 차선인식 및 주행 알고리즘 구현 ○ 회전 교차로 알고리즘 구현 ○ 신호등 분기점 알고리즘 구현
3	팀원	박상재	○ 주차 미션 구현 ○ 차로 추월 구간 알고리즘 구현 ○ 주행 알고리즘
4	팀원	이진호	○ 알고리즘 통합 ○ 차선인식 및 주행 알고리즘 구현 ○ 터널 알고리즘 구현
5	팀원	고승일	○ 차선인식 및 주행 알고리즘 구현 ○ 개발 환경 구축