제16회 임베디드SW경진대회

개발완료보고서 [자율주행 모형자동차]

1. 작성 시 주의사항

※ 제출 분량 : A4 용지 상세내용 포함 20 page 이내 (최대 폰트 크기 12pt)
※ 제출 포맷 : pdf

2. 팀 정보

팀명	FOSCAR	팀장	국민대학교 박희승
팀원	국민대학교 김철현	팀원	국민대학교 이한정
팀원	국민대학교 전재신	팀원	국민대학교 현혜림

3. 본 개발완료보고서

0. 작품 제목

1. 개요

1.1. 작품 개요

1.2. 필요성

1.3. 개발 목표

2. 작품 설명

2.1. Software 구성

2.2. Software 설계도

2.3. Software 기능

2.4. 프로그램 사용법

2.5. 개발환경

3. 프로그램 설명

3.1. 파일 구성

3.2. 함수별 기능

3.3. 주요 함수의 흐름도

3.4. 기술적 차별성

4. 개발 중 장애요인과 해결방안

5. 개발결과물의 차별성

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

0. 작품 제목

나누의 꿈

1.1. 작품 개요

이 작품은 임베디드 보드를 이용한 자율주행 자동차가 스스로 상황을 판단하여 다양한 상황들에 대처하는 것에 의의가 있다. 차량이 주행하는 동안 차선 유지 및 변경, 주차, 장애물 인식, 신호등 및 표지판 인식 등을 자율적으로 수행할 수 있도록 임베디드 소프트웨어를 개발하고, 더 나아가 실제 도로 상황에서 발생할 수 있는 문제들에 대해 정확히 판단하고 대처할 수 있도록 구성되어 있다.

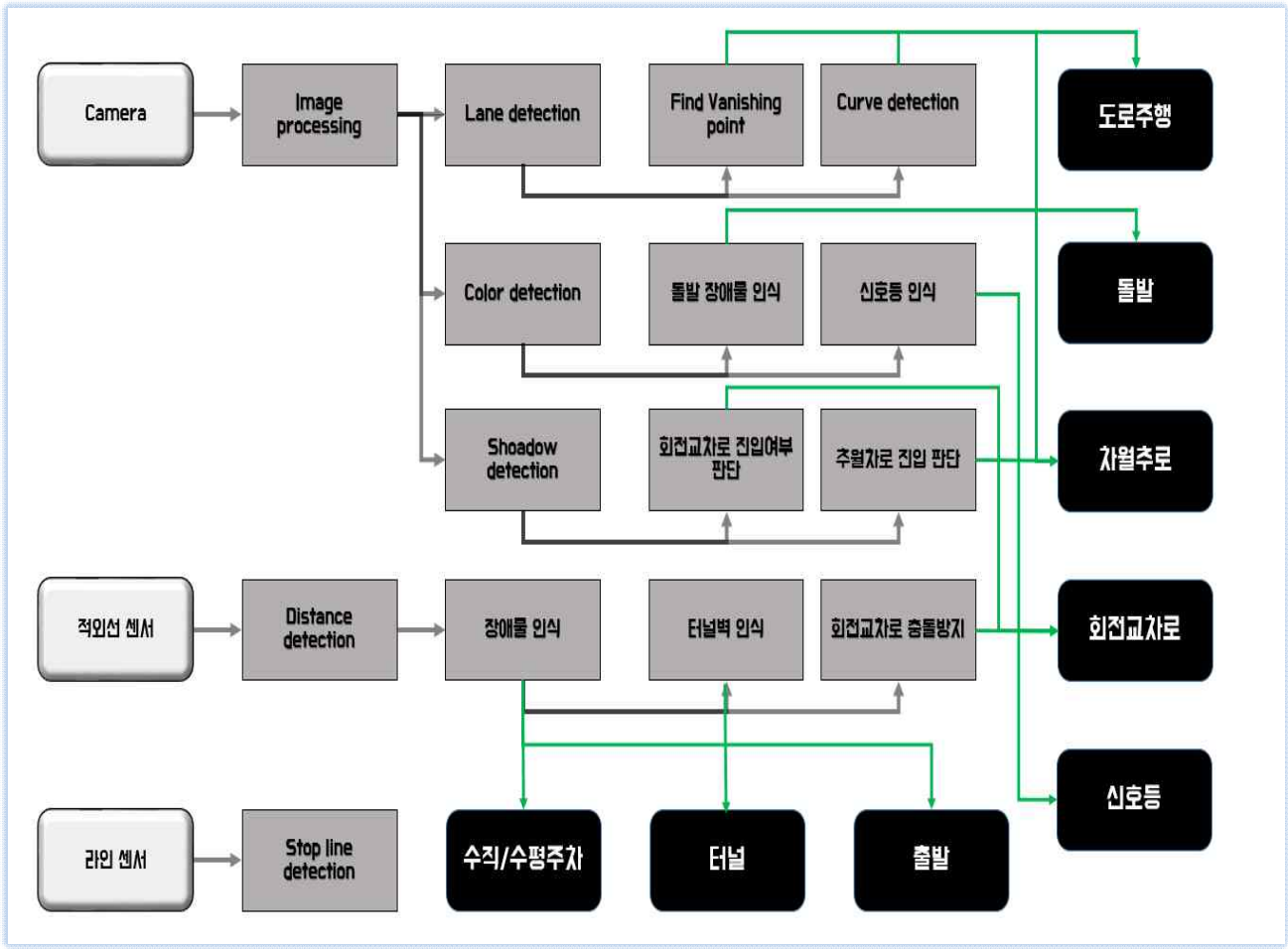
1.2. 필요성

자율주행 자동차는 운전자의 편의를 제공할 수 있고, 남녀노소를 비롯해 장애인도 운전자가 될 수 있어 많은 문제를 해결할 수 있다는 점에서 사회적으로 큰 이슈가 되고 있다. 자율주행 자동차의 기술의 발전을 위해 다양한 상황을 모형으로 재현하여 시뮬레이션하고, 더 나아가 실제의 도로나 차량 등 여러 환경에서 자율주행 자동차가 적응할 수 있도록 다양한 알고리즘을 구현할 필요성이 있다.

1.3. 개발 목표

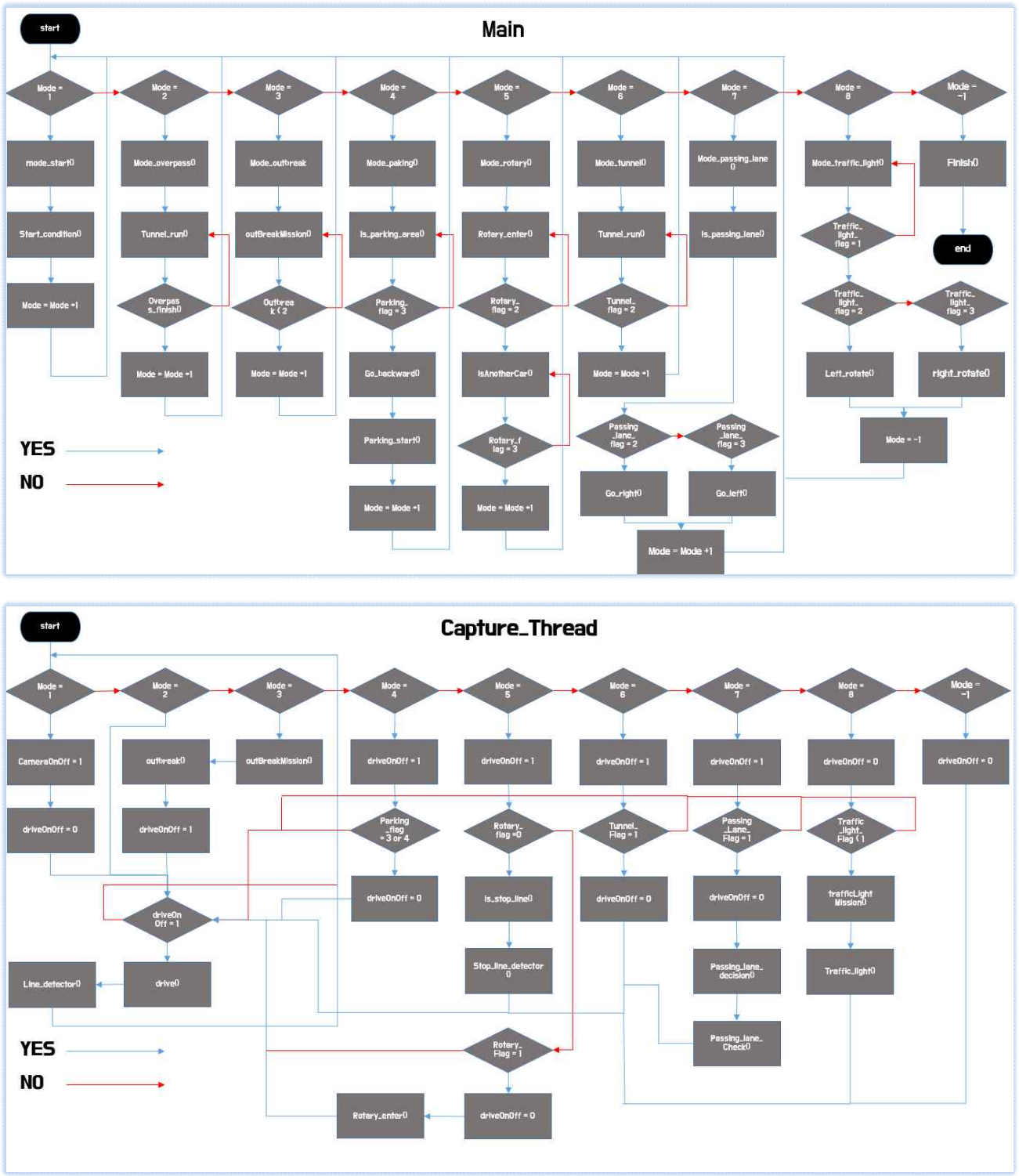
자율주행 모형자동차 부문은 실제 자동차가 주행하는 것과 비슷하게 모형차로 자율주행을 시뮬레이션하는 것이 목적이다. 실제 자동차에 적용할 수 있도록 실시간성을 갖추기 위해 모든 연산은 간결화되어야 한다. 따라서 필요한 연산들만 수행하여 소프트웨어를 간결하게 구성하고, 어느 환경에서든지 동일하게 주행할 수 있도록 외란에 강한 능력을 갖춘 자율 주행 자동차를 만들 것이다. 또한 차량에 올라가는 임베디드 보드의 특성상 한정된 자원으로 좋은 성능을 내야 하므로 효과적인 알고리즘을 구현하여야 한다.

2.1. Software 구성



Camera, 적외선 센서, 라인 센서를 통해 영상 및 센서 값을 받아오고 Processing을 통해 미션을 수행한다.

2.2. Software 설계도



쓰레드는 Capture Thread와 Main 총 2개로 구성 되어 있다. Capture Thread는 카메라로 부터 획득한 영상을 버퍼에 담아 순차적으로 Image Processing한 후 LCD에 그 영상을 띄워준다. 반면 Main은 Capture Thread로부터 제어 값 또는 결과 값을 받아 미션별로 제어를 한다.

미션의 구분은 mode라는 변수를 사용하여 미션이 끝나면 mode를 1을 더해주며 수행한다.

2.3. Software 기능

2.3.1 차선인식 및 도로주행

1. 전처리

1) 관심영역 설정

영상처리의 연산량을 줄이고 불필요한 영역을 제거하기 위해 관심영역(ROI)을 설정한다. 관심영역의 크기는 상황에 따라 가변적으로 적용한다. 또한 차선의 영역을 더 많이 획득하기 위해 카메라 각도를 내린다.

2) 이진화

효과적인 차선 검출을 위해 이진화 단계를 거친다. HSV로 변환한 영상에서 노란색부분의 픽셀을 지정한 범위에 따라 이진화를 진행한다. 상황에 따라 흰색차선도 이진화를 진행한다.

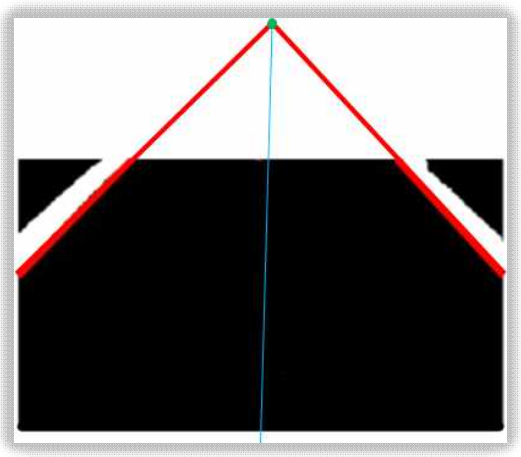
전처리 과정을 거치면 차선 검출을 시작한다.

2. 차선 검출

Canny Edge 검출 알고리즘 통해 이진화된 영상에서 edge를 검출한 후 Hough Transform 알고리즘을 적용하여 차선의 후보군을 검출한다. 많은 후보군 중에서 대표차선을 뽑아내야 하는데 K-means 알고리즘을 적용하여 후보군을 군집화하고 그 군집화된 영역의 중심을 구해서 최적의 차선을 찾아낸다.

3. 직선 주행

1) 소실점 제어



기본적으로 좌우 차선이 모두 검출되면 소실점을 검출한다. 현재 차의 중심 위치는 카메라 영상의 중심(160)과 소실점의 x값에 따른 조향 값을 얻어 제어한다.

4. 곡선 주행

1) 기울기, 치우침 제어

한쪽 차선이 인식 안 될 경우 곡선 코스라 가정을 한다. 이때는 소실점이 아닌 검출된 한 쪽 차선의 기울기와 차선과의 치우침 값으로 제어를 한다. 치우침 값을 얻을 때는 검출된 차선에 ROI에 따른 y값을 넣어 구한다. 기울기에 따른 조향 값을 얻은 후 치우침 값을 가중치로 곱해주어 부드러운 코너링을 할 수 있도록 한다.

2.3.2 돌발장애물 인식

1) 장애물 인식

정상 주행 중 카메라를 통해 들어오는 영상의 프레임마다 전체영역에서 HSV로 변환하여 적색 돌발 장애물을 지정한 범위에 따라 이진화한 후 이진화된 픽셀의 개수를 센다.

2) 판단

설정한 임계치를 넘어가는 프레임이 검출되면 차를 즉시 정지시킨다. 정지한 상태에서 계속 이진화 된 픽셀의 개수를 검출하고 임계치 이하가 되면 돌발 장애물이 완전히 사라질 때까지 일정 시간 기다린 후 다시 원래대로 정상 주행을 수행한다.

2.3.3 자동주차

1) 주차공간 판단

측면 거리 센서를 이용하여, 물체와의 거리를 측정하면 1차 장애물을 인식하면서 거리가 인식 거리가 짧아지고, 주차공간을 인식하면서 늘었다가 다시 2차 장애물을 인식하면서 짧아지게 되는 구간이 생기는데, 이를 주차공간을 인식하고 주차 미션을 수행한다.

2) 조향 값 결정

주차 공간을 인식하게 되면, 주차 공간에 진입하기 위해 조향 값을 결정해야 하는데, 장애물과의 거리 값의 범위를 조향 값의 범위로 치환시켜 조향 값을 결정한다.

3) 수직주차와 수평주차의 구분

차량이 주차공간에 진입하게 되면 1차 장애물까지의 거리를 측정하여 거리가 20cm보다 작으면 수직주차 미션으로 인식하고, 크면 수평주차 미션으로 인식하여 각각의 주차함수를 실행한다.

1. 수직주차

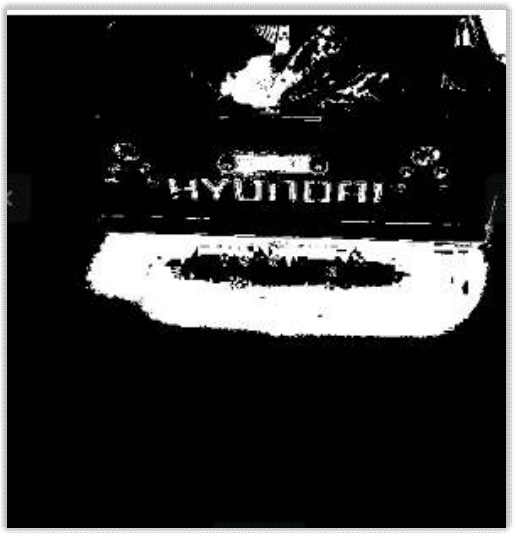
주차공간 진입 후, 수직주차로 인식하게 되면 1차 장애물과 2차 장애물까지의 거리를 각각 측정하여 비교하면서 거리가 더 큰 쪽으로 조향을 하면서 진입하게 된다. 수직주차가 완료되면, 주차공간을 완전히 탈출하고, 차선을 인식할 때까지 우측 조향하면서 진행하고, 차선 인식이 되면 차선 주행을 시작한다.

2. 수평주차

주차공간 진입 후, 수평주차로 인식하면 주차를 실행하기 위한 공간을 벌기 위해 앞으로 전진했다가 측면 거리 센서와 후방 거리센서의 인식 거리를 바탕으로 조향값을 조정하면서 주차를 완료하게 된다. 수평주차가 완료되면, 왼쪽 조향을 하면서 주차공간을 탈출하고 차선 인식이 되면 차선주행을 시작한다.

2.3.4 회전교차로 주행

1) 회전교차로 진입 판단



정지선 인식 후 대기하다가 회전교차로 주행 중인 차량이 지나가고 차량의 그림자를 인식하고 출발한다. 출발 시 속도는 낮게 유지한다.

2) 회전교차로 주행

천천히 주행하다가 주행 중인 차량이 뒤에 가까이 붙으면 속도를 높여서 주행하기 시작한다.

3) 회전교차로 탈출

뒤에 있던 차량이 어느 정도 멀어진 것이 판단되면 회전교차로 구간이 끝났다고 인식한다.

정지선 인식은 차량 바닥 쪽에 붙어있는 라인센서를 이용하여 인식했다. 대기 중에 지나간 차량의 유무 판단을 할 때는 그림자를 이용하는데 읽어 들이는 ROI(관심영역)를 설정하고 영상을 YUV로 변환하여 어두운 부분을 지정한 범위에 따라 이진화 한다. 값이 1인 픽셀들의 개수가 임계치 이상이면 차량의 그림자라고 판단한 후 임계치 이하로 내려가면 출발한다.

앞, 뒤에 주행 중인 차량의 유무를 판단할 때에는 적외선 센서를 이용한다. 센서에 들어오는 값이 일정 값 이하가 되면 차량이 각각 앞, 뒤에 있다고 판단하여 속도를 조절한다.

2.3.5 고가도로 및 터널주행

1) 터널 진입 및 탈출

터널 진입 판단은 좌, 우 앞쪽의 적외선 센서의 값이 모두 일정 값 이하가 되면 터널의 벽이라고 인지하고 터널 주행모드를 시작한다. 마찬가지로 탈출할 때도 적외선 센서를 이용한다. 이때는 반대로 위와 같은 센서의 값이 모두 일정 값 이상이 되면 벽이 사라진 거로 인식하고 터널 주행 모드를 끝낸다.

2) 터널 주행

터널 내부가 어둡기 때문에 적외선 거리 센서를 이용하여, 주행하게 된다. 왼쪽 거리 센서와 오른쪽 거리 센서의 인식거리를 비교하여, 더 큰 쪽으로 조향을 하게 되면, 두 센서의 인식거리가 같아지는 알고리즘을 이용하여, 터널 주행을 한다.

2.3.6 추월차로 주행

1) 추월차로 진입 판단

먼저 적외선으로 정면에 장애물이 있는지 판별한다. 일정 거리 이내에 장애물이 판별되면 양쪽 차선 중 어느 차선에 장애물이 있는지 판별한다. 판별 시에는 장애물 차량이 만든 그림자를 YUV색공간으로 나타낸 이미지에서 이진화하여 픽셀 수를 계산한다. 차량이 만든 그림자가 어떤 차가 만든 그림자인지 판별할 때에는 간단한 부등식을 사용한다. 흰색 차선을 검출하고, 그 차선이 만들어낸 직선의 방정식을 대입하여 이진화된 화면에서 나온 좌표를 부등식에 대입하여 판별하게 된다.

2) 추월차로 주행

차선 변경 시에는 비어있다고 판단된 차선 쪽으로 조향한 뒤 다시 차선 인식을 해서 진행하게 된다. 가운데 차선으로 복귀 시에는 정면 쪽에 카메라 ROI를 설정해 두고 노란색 차선의 기울기가 0에 가깝게 검출된 경우 가운데 차선 쪽으로 조향하고 차선인식을 시작한다. 마지막으로 라인센서가 흰색 선을 인식하게 되면 정지한 후 신호등 주행을 시작한다.

2.3.7 신호등 분기점 주행

1. 전처리

신호등 이외의 부분에서 빨강, 노랑, 초록을 검출하면 알고리즘에 문제가 생기기 때문에 관심영역을 설정하여 그 외 부분을 연산하지 않도록 한다.

빨강, 노랑, 초록 세 가지의 HSV 색 범위에 맞게 영상을 이진화하여 해당 색의 검출이 필요할 때, 원하는 색의 HSV 범위를 이용하여 픽셀 값을 얻는다.

2. 판단

세 가지 색으로 이진화된 영상을 이용하여 빨간색 등이 들어올 때의 픽셀 값들의 중앙값, 노란색 등이 들어올 때의 픽셀 값들의 중앙값을 이용하여 2개의 초록색 등 좌표(좌회전, 우회전)의 중심을 추측하여 구한 뒤, 초록색 등이 들어올 때, 초록색 등의 픽셀 값들의 중앙값과 비교하여 좌·우측 조향을 판단한다.

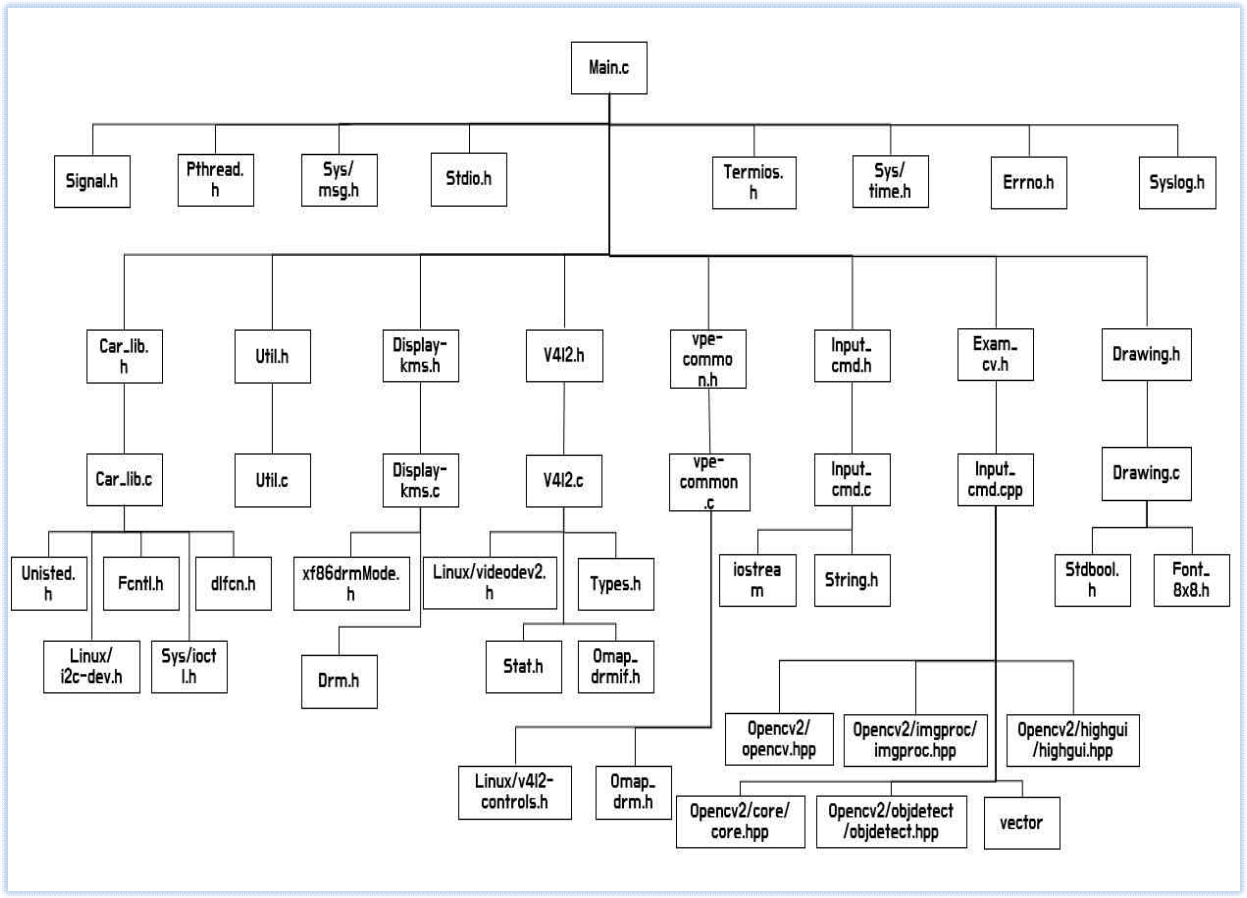
2.4. 프로그램 사용법 (Interface)

순서 : (@PC) 컴파일 (@PC) target board로 바이너리 전송 (@보드)전송된 바이너리 실행

2.5. 개발환경 (언어, Tool, 사용시스템 등)

개발환경	Linux OS(Ubuntu16.04) Minicom NFS 서버 Eclipse CDT 에디터 Github
통신	Ethernet 통신 , Serial 통신, 적외선 통신 (GP2Y0A21F03)
사용 언어	C/C++
자동차 프로세서	TI (Texas Instruments) TDA2x (ARM Cortex-A15 Dual core 1.5GHz)

3.1. 파일 구성



3.2. 함수별 기능

3.2.1 drive()

drive()함수는 차선인식과 동시에 서보모터를 제어하는 함수이다. 상황에 따른 line_detector()함수를 실행하여 조향 값을 얻어 서보모터를 제어한다.

3.2.2 mode_start()

출발 신호를 주기 위한 함수이다.

차가 정지된 상태에서 출발하기 위해 1번(정면) 적외선 센서를 이용해 출발 신호를 주었다. 출발 신호가 떨어지면 mode를 증가시켜 mode_overpass()를 호출할 수 있도록 하고 mode_start()는 종료된다.

3.2.3 mode_overpass()

고가도로 구간을 주행하기 위한 함수이다.

출발하면 바로 고가도로 모드가 실행된다. 고가도로 모드가 실행되면 overpass_finish() 함수를 이용해 현재 주행하고 있는 곳이 고가도로인지 확인을 한다. 이 때 확인은 2번, 6번(우, 좌) 적외선 센서를 이용하여 외벽이 존재하면 고가도로로 인식을 하고 외벽이 사라지면 고가도로가 아닌 거로 판단을 하여 mode_outbreak()을 호출하게끔 mode를 1만큼 증가시키고 mode_overpass()는 종료된다.

3.2.4 mode_outbreak()

우선 정지 장애물 구간을 위한 함수이다.

고가도로에서 내려온 뒤 카메라로부터 받아온 영상에서 빨간색 픽셀 수를 세어 우선 정지 구간인지 아닌지를 판단한다. 픽셀의 수가 임계치 이상이면 장애물을 인식한 것으로 판단하고 outbreak_flag = 1로 값을 바꾼다. 다시 픽셀의 수가 임계치 이하로 떨어지면 장애물이 사라진 것으로 판단하여 outbreak_flag = 2로 수정한다. 이때, mode_parking()을 호출하게끔 mode를 1만큼 증가시키고 mode_outbreak()는 종료된다.

3.2.5 mode_parking()

자동주차를 실행하기 위한 함수이다.

주차공간 전에 위치한 S자 곡선 주행을 완료했는지 판단을 하고 주차공간을 찾기 시작한다. 이는 S자 곡선 주행 중에 잘못 주차 공간을 인식하는 것을 막기 위함이다. 적외선 센서를 이용하여 첫 번째 주차공간 벽을 인식하면 parking_flag = 1로 바꾸고 주차공간을 인식하면 parking_flag = 2로 마지막으로 두 번째 주차공간 벽을 인식하면 parking_flag = 3으로 바꾼다.

parking_flag = 3이 되면 go_backward() 함수를 호출하여 뒤의 벽까지와의 거리가 40이 될 때까지 후진 주차를 시작한다. 그리고 3번, 5번 적외선 센서를 사용하여 양옆의 공간을 판단하여 공간이 좁으면 수직주차를 하고, 공간이 충분하면 수평주차를 한다.

주차가 끝난 뒤에 차선으로 복귀하기 위해 수직, 수평 주차에 대해 각각 return_lane_vertical(), return_lane_horizontal() 함수를 호출하여 복귀한다. 한 번의 주차가 끝난 뒤 주차가 한 번 더 있기 때문에 한 번의 완료를 표시하기 위해 parking_finish = 1로 바꾸고 parking_flag는 다시 0으로 초기화하여 위의 과정을 반복한다.

두 번째 주차까지 모두 완료하면 mode_rotary()를 호출할 수 있도록 mode를 1만큼 증가시키고 mode_parking()은 종료된다.

3.2.6 mode_rotary()

회전 교차로 구간을 주행하기 위한 함수이다.

차선인식을 통해 주행하다가 정지선이 발견되면 멈추고 rotary_flag = 1로 바꾸고 주행 중인 차량이 앞을 지나갈 때까지 대기한다. 차가 앞쪽으로 지나가는 것은 영상을 이진화하여 그림자를 이용하여 판단한다.

차가 지나가면 rotary_flag = 2로 바꿔주고 낮은 속도로 출발하여 교차로 구간을 주행한다. 교차로 구간을 주행하다가 주행 중인 차가 뒤쪽에 가까이 오면 속도를 높여 빠르게 주행하고 rotary_flag = 3으로 바꾸면서 이때 mode를 증가시켜 mode_tunnel()을 호출하고 mode_rotary()는 종료된다.

3.2.7 mode_tunnel()

터널 구간 주행을 위한 함수이다.

mode_tunnel()이 실행되면 차선인식을 하다가 2번, 6번 적외선 센서를 이용하여 양 옆의 벽을 인식하면 터널 진입 구간으로 판단한다. 터널 구간이 시작되면 카메라는 끄고 적외선 센서만을 이용하여 주행한다. 좌우로 얼마만큼 떨어졌는지 확인하고 가운데로 갈 수 있도록 tunnel_run()함수를 구현하였다.

주행하다가 적외선 센서로부터 받아온 값이 임계치 이상이면 벽이 없어졌다고 판단하여 터널 종료 구간으로 인식한다. 이때 mode를 1만큼 증가시켜 mode_passing_lane()을 호출하고 mode_tunnel()은 종료된다.

3.2.8 mode_passing_lane()

차로 추월 구간을 주행하기 위한 함수이다.

우선 mode_passing_lane()이 실행되는 mode값일 때에는 흰색과 노란색을 모두 이진화하는 차선인식 코드가 실행된다.

mode_passing_lane() 함수가 실행되면 먼저 적외선으로 정면에 장애물이 있는지 판별한다. 일정 거리 이내에 장애물이 판별되면 양쪽 차선 중 어느 차선에 장애물이 있는지 판별한다. 판별 시에는 장애물 차량이 만든 그림자를 YUV색 공간으로 나타낸 이미지에서 이진화하여 픽셀 수를 계산한다. 차량이 만든 그림자가 어떤 차가 만든 그림자인지 판별할 때에는 간단한 부등식을 사용한다. 흰색 차선을 검출하고, 그 차선이 만들어낸 직선의 방정식을 대입하여 이진화된 화면에서 나온 좌표를 부등식에 대입하여 판별하게 된다. 차선 변경 시에는 비어있다고 판단된 차선 쪽으로 조향한 뒤 다시 차선인식을 해서 진행하게 된다.

가운데 차선으로 복귀 시에는 정면 쪽에 카메라 ROI를 설정해 두고 노란색 차선의 기울기가 0에 가까워질 때 가운데 차선 쪽으로 조향하고 차선인식을 시작한다. 마지막으로 라인센서가 흰색 선을 인식하게 되면 정지하고 mode를 1만큼 증가시켜 mode_traffic_light()를 호출하고 mode_passing_lane()은 종료된다.

3.2.9 mode_traffic_light()

신호등 구간을 통과하기 위해 구현한 함수이다.

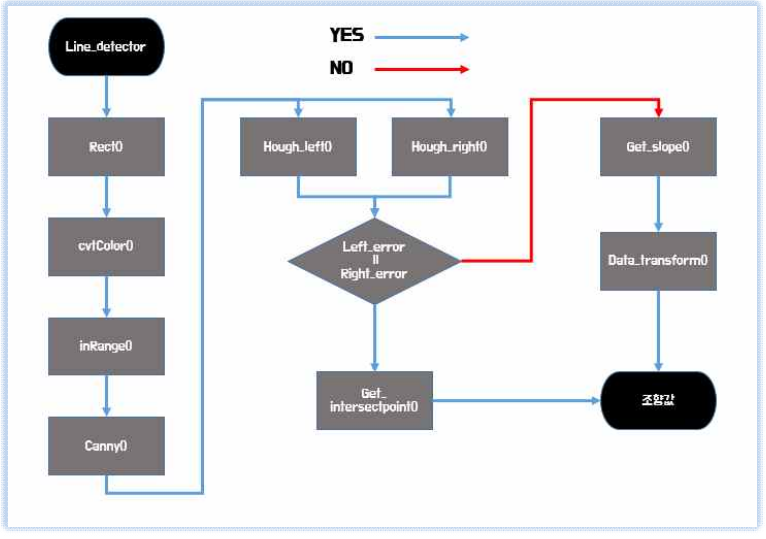
우선 시작 부분인 흰색 정지선에서 카메라를 이용하여 신호등을 본다. 신호등은 빨강, 노랑, 초록 순으로 점등하게 되는데, 신호 3가지가 모두 켜져있는 것이 아니고 하나씩 차례로 켜졌다가 꺼지기 때문에, 각각의 신호가 켜졌을 때, 카메라에서 점등된 좌표를 저장하게 해놓았다. 저장한 빨강과 노랑의 좌표를 이용하여 좌회전 신호, 우회전 신호를 구별하였다.

빨강과 노랑의 좌표 값 중 가로의 차이를 x 라 할 때, 빨강에서 x의 2.5배만큼 이동하게 되면 좌회전 신호와 우회전 신호의 중간 지점의 좌표를 계산할 수 있다. 빨강과 노랑의 점등으로 여기까지 계산한 뒤에 초록색이 점등하는 즉시, 계산한 좌표보다 좌측에서 점등할 시 좌회전, 우측에서 점등할 시 우회전하게 하였다. 초록색이 점등되면 일단 직진을 하게 한다.

신호등과의 거리가 가까워지는데, 이때 특정한 값 이하로 가까워지게 되면 빨강과 노랑을 이용하여 계산된 진입 구간으로 최대 조향을 하여 진입하게 된다. 최대 조향 중 노란색 차선의 기울기를 이용해서 직진해도 된다고 판단되게 되면 라인 센서가 흰색을 발견할 때까지 직진하게 된다. 라인센서가 흰색을 검출하게 되면 usleep()을 이용하여 몇 초간 직진을 하다가 주행 완료를 출력한 뒤 주행을 종료한다.

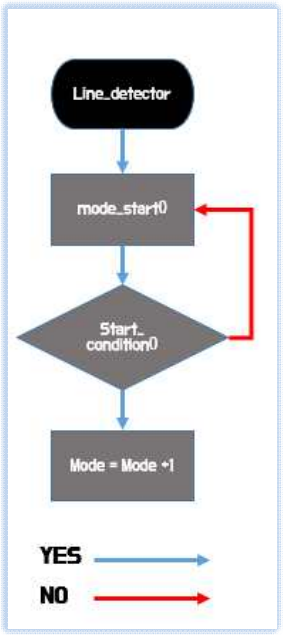
3.3. 주요 함수의 흐름도

3.2.1 line_detector()



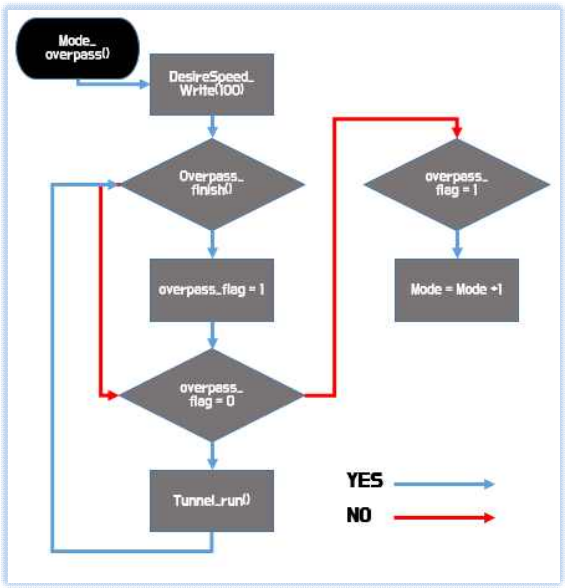
line_detector()는 차선인식을 하여 조향 값을 return해주는 함수이다. 처음 Rect() 함수를 사용하여 ROI설정을 한다. 그 후 hsv변환 후 inRange()함수를 사용하여 이진화를 진행한다. 전처리 과정이 끝나면 canny()함수를 사용하여 엣지를 검출한 후 좌, 우 차선을 Hough_left(), Hough_right() 함수를 이용해 각각 검출한다. 그 후 Get_intersectpoint()함수를 사용하여 교점을 구해 조향 값을 얻는다.
만약 Left_error 또는 Right_error가 발생하면 기울기와 치우침을 구해 조향 값을 얻는다.

3.2.2 mode_start()



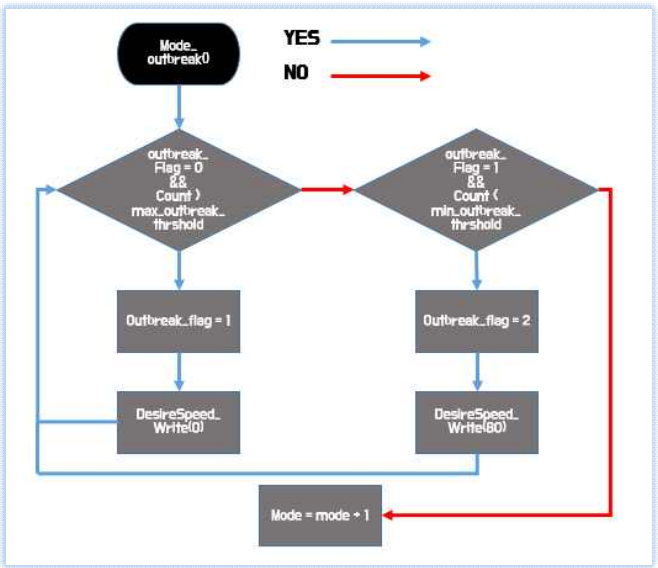
출발 신호를 확인하기 위한 함수이다. mode_start()가 시작되면 start_condition()을 호출하여 정면에 있는 적외선 센서로부터 값을 판단해 값이 작아졌다 커지면 출발하라는 신호로 전달받는다. start_flag가 0인 상태에서 거리가 10보다 작아지면 flag를 1로 만들고, 1인 상태에서 20보다 커지면 flag를 2로 만들고, flag가 2일 경우에는 true를 반환한다. start_condition()에서 true를 반환하면 출발하라는 신호로 인지하고 주행모드를 시작한다.

3.2.3 mode_overpass()



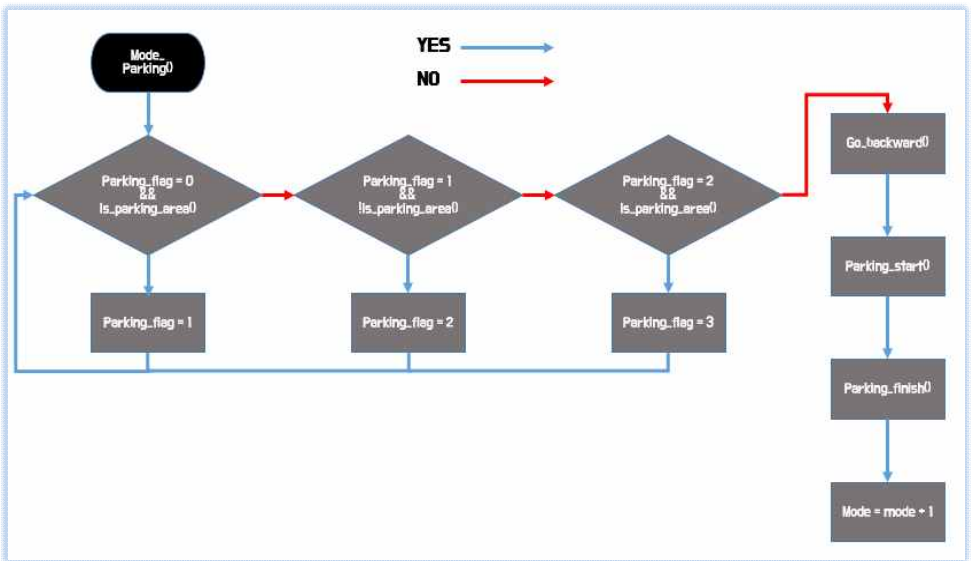
출발신호를 받고 좌, 우 앞쪽 적외선 센서 값이 일정 범위 안으로 들어오면 고가도로의 외벽이라고 인식하여 고가도로 모드가 시작된다. 고가도로를 주행하는 알고리즘은 터널과 마찬가지로 양쪽에 외벽이 존재하기 때문에 터널 주행과 같은 알고리즘을 사용한다. 좌, 우 앞쪽 적외선 센서의 값을 읽어들여 그 차이를 가지고 해당 조향값을 입력한다. 또한 주행 중 overpass_finish()함수를 호출하여 좌측 앞쪽 적외선 센서의 값이 임계치 이상으로 커지면 고가도로의 외벽이 없어진 것으로 판단하여 고가도로 주행을 마친다.

3.2.4 mode_outbreak()



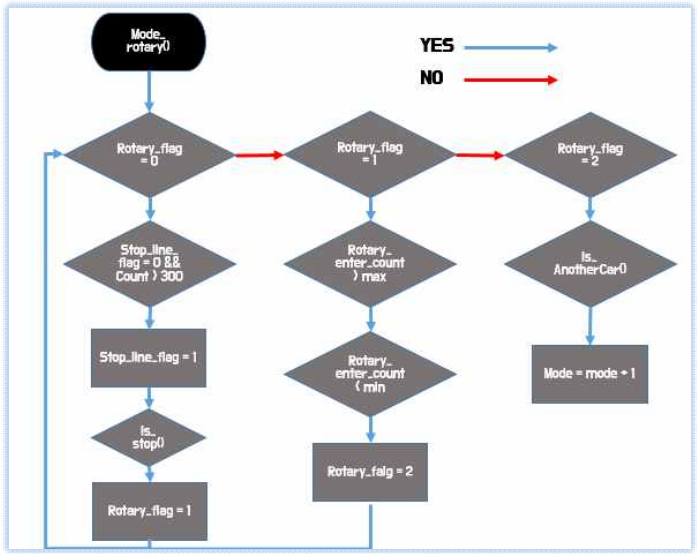
설정한 임계치를 넘어가는 프레임이 검출되면 outbreak_flag를 1로 바꿔준 후 차를 정지시킨다. 그 후 임계치보다 낮아지면 outbreak_flag를 2로 바꿔주고 속도를 다시 올려준다. 그 후 mode를 1 올려준다.

3.2.5 mode_parking()



주차 구간을 판단하기 위해서 가장 중요한 것이 is_parking_area()라는 함수인데, 1차 장애물, 주차 공간 그리고 2차 장애물의 순서로 인식된 다음, parking_flag가 3일 때 주차구간으로 인식하고 그 외의 경우에는 다시 초기화하여 주차 공간 인식을 한다. 주차 구간이 인식되면 go_backward() 함수가 실행되는데, 이는 조향값을 결정하고 주차공간으로 진입하기 위한 함수이다. parking_start() 함수는 주차 공간의 너비에 따라 수평주차와 수직주차를 구분하게 되고, 임계값을 기준으로 크면 수평주차로 판단하고 작으면 수직주차로 판단하여 각각의 vertical_paring() 함수와 horizontal_parking() 함수를 실행한다.

3.2.6 mode_rotary()

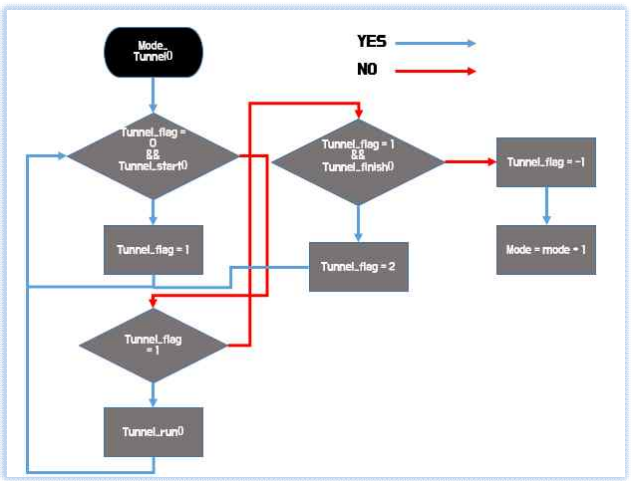


회전 교차로 모드가 실행되면 제일 먼저 정지선을 인식한다.

is_stop_line()함수로부터 얻은 stop_line_count의 갯수가 임계치 이상이면 정지선으로 인식하고 교차로에서 주행 중인 차량이 지나가길 기다린다. rotary_enter() 함수로부터 rotary_enter_count의 값으로 차량의 그림자 유무를 확인한다. 앞차가 지나가면 낮은 속도로 주행을 시작한다.

주행 중에 isAnotherCar() 함수를 이용해 뒤쪽에 차량의 유무를 확인하고, 만약 뒤에 다른 차량이 가까이 온 것이 감지되면 속도를 높여 주행을 하여 교차로를 탈출한다

3.2.7 mode_tunnel()

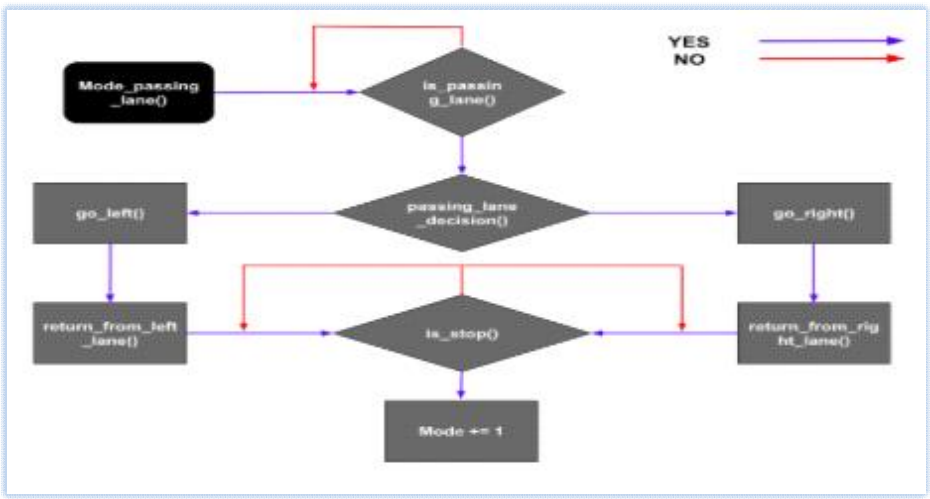


mode_tunnel()이 시작되면 좌, 우 앞쪽 적외선 센서로부터 값을 읽어들인다. 만약 두 센서로부터 읽어들인 값이 모두 임계치 이하가 되면 터널 진입 구간으로 판단하고 tunnel_run()함수를 호출해 주행을 진행한다.

tunnel_run() 은 왼쪽만 인식할 때, 오른쪽만 인식할 때, 둘 다 인식할 때로 나뉘는데 왼쪽만 인식할 때는 왼쪽 벽을 따라가도록 하고, 오른쪽만 인식할 경우에는 오른쪽을 따라가도록 조향값을 주었다. 만약 둘 다 인식한다면 좌, 우의 값을 비교해서 값을 차이에 비례하여 조향값을 주어 주행하였다.

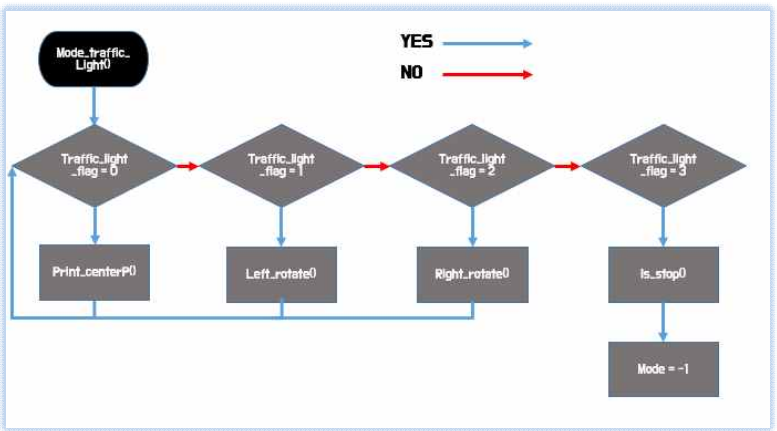
만약 양쪽 적외선 센서의 거리가 모두 임계치 이상이면 터널의 탈출 구간으로 인식하고 터널 주행 모드를 종료한다.

3.2.8 mode_passing_lane()



Mode_passing_lane()함수가 실행되면 먼저 is_passing_lane() 함수를 호출한다. is_passing_lane()함수는 YUV 임계값으로 이진화된 화면에서 일정 임계값 이상의 픽셀수가 인식되면 passing_lane_flag값이 1이 되고 어느 차선으로 가야하는지 판별하는 passing_lane_decision()함수가 실행된다. passing_lane_decision() 함수가 왼쪽 차선이 비었다고 판별하면 go_left()를 실행하고, 오른쪽 차선이라고 판단하면 go_right() 함수를 실행한다. go_left()와 go_right()함수를 실행 중에 정면 부분에 노란색 차선이 인식되면 추월 차로가 끝났다고 판단하고, 각각 return_from_left_lane() 함수와 return_from_right_lane() 함수를 실행한다. 그리고 마지막으로 is_stop() 함수를 실행하여 라인센서가 인식되면 Mode를 1 증가시켜 traffic_light 모드로 넘어가게 된다.

3.2.9 mode_traffic_light()



Mode_traffic_Light() 함수는 Traffic_light_flag = 0일 때, TrafficLightMission()함수를 이용하여 카메라의 영상정보를 입력받아 신호등의 판단을 수행한다. 판단이 안 될 시 traffic_light_flag는 0이며 계속 TrafficLightMission()을 호출하게 된다. 초록색의 픽셀을 발견하여 신호등의 좌우회전을 판단하였을 때, 좌회전으로 판단할 시 traffic_light_flag는 1이 되며 좌회전함수를 실행하고 , 우회전으로 판단할 시 traffic_light_flag는 2가 되며 우회전함수를 실행한다. Traffic_light_flag = 3일 때, is_stop() 함수가 실행되며, is_stop()에서는 정지선을 인식한 후, traffic_light_flag를 4, finish_flag를 1로 만든 다음, usleep()을 이용하여 일정시간 직진하다 정지하고, "...finish..." 라는 메시지를 출력한 뒤, 주행을 종료한다.

3.4. 기술적 차별성

1) 연산량을 줄이기 위한 V_ROI(가변적 ROI)를 사용하여 연산량을 크게 줄였다.

2) 각 미션별 모듈화를 진행하여 원하는 미션의 테스트 mode변수만 바꿔주어서 바로바로 진행 할 수 있도록 하였다.

3) 추월차로에서 차선인식을 하여 영역을 구분하여 어느 위치에 정확히 차량이 존재하는지 판별하였다.

4) 여러 가지 센서를 사용함에 있어서 여러 가지 단위들이 파라미터로 전달되는데 그 파라미터들을 원하는 단위로 변환하여 정확한 값을 전달할 수 있다.

4. 개발 중 장애요인과 해결방안

도로 주행	문제점 : 곡선주행의 경우 차선의 기울기로만 제어를 하면 부드럽지 못하고 주행중 차선 이탈이 빈번히 발생하였다. 해결책 : 차선과 차량의 치우침을 계산하여 제어 값의 가중치를 곱해주었다. 그 결과 부드러운 곡선 주행을 하였다.
주차 구간	문제점 : 다른 구간의 장애물로 인해 주차구간이 아닌 미션을 주차구간으로 오인하는 경우가 생겼다. 해결책 : 거리 센서값을 cm단위로 변환하여, 실제 거리 값에 근접한 값으로 인식하고, 주차 구간의 정규 규격에 맞추어 주차구간을 인식할 수 있도록 하였다
회전 교차로	문제점 : 회전 교차로 탈출 구간을 판단하는 조건이 확실하지 않아서 구체적인 방법이 필요했다. 해결책 : 앞에 있던 차량이 뒤에 따라붙어 속도가 높아지는 경우에 교차로의 마지막 부분이어서 그때 탈출을 했다고 인지했다.
터널 구간	문제점 : 터널 진입 구간을 영상처리를 이용하여 판단하였지만, 좀 더 정확한 판단 방법이 필요했다. 해결책 : 측면 거리센서를 이용하여, 양쪽 측면 거리센서에 작은 인식 거리 값이 들어오면, 터널 구간 진입으로 인식하도록 구현하였다.
추월 차로	문제점 : 진입 방향을 판단할 때 단순히 영상을 3등분하면 주행 방향에 따라 정확한 판단이 불가능하였다. 해결책 : 흰색 차선을 기준으로 영역을 분할하여 정확한 차선 영역을 통해 정확한 판단을 하였다.
신호등	문제점 : 추월 차로 종료 후 차가 일자로 정지하지 못하여 신호등 인식 후 조향할 때 일정한 조향 값으로 할 시 문제가 발생하였다. 해결책 : 카메라로 신호등 앞에 있는 노란색 선의 기울기를 검출하여 기울기에 따른 조향 값과 usleep()값을 주었다.

5. 개발결과물의 차별성

1) 연산량을 줄이기 위한 V_ROI(가변적 ROI)를 사용하여 연산량을 크게 줄였다.

2) 각 미션별 모듈화를 진행하여 원하는 미션의 테스트를 바로바로 진행 할 수 있도록 하였다.

3) 추월차로에서 차선인식을 하여 영역을 구분하여 어느 위치에 정확히 차량이 존재하는지 판별할 수 있다.
또한, 추월차로에 2대의 차량이 있어도 빈 차로를 판단해 주행이 가능하다.

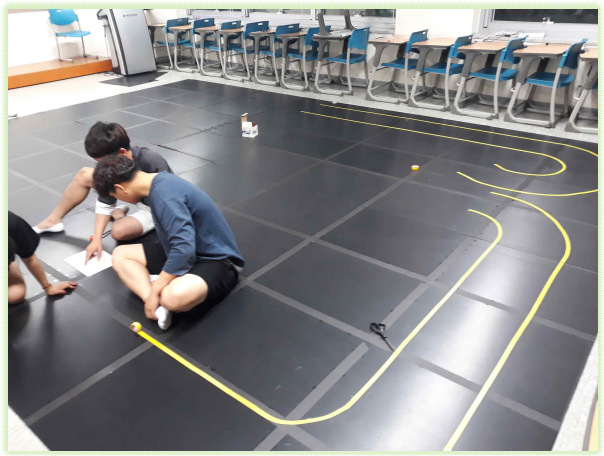
4) 화면에 이진화된 영상을 띄워 빠른 디버깅을 할 수 있게 하였다.

5) 터널 주행 같은 경우에도 한쪽 벽만 존재 하여도 주행이 가능하다.

6) 함수의 모듈화가 잘 이루어졌기 때문에 빠른 디버깅과 코드 재활용이 가능하다. 실제로 모듈화된 함수가 여러 미션에서 사용되고 있다. ex) drive(), isStop(), Datatransform() 등등

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

2018.7~8 트랙제작




포맥스와 기타 재료를 이용하여 실제 트랙과 유사한 환경을 구축하여 시험 경기장으로 사용하였다.

2018.8~10 개별적 모듈개발 및 디버깅
2018.9~10 통합 및 전체적인 테스트 및 디버깅

No.	구분	성명	학교	담당
1	팀장	박희승	국민대학교	영상처리 및 통합
2	팀원	김철현	국민대학교	주차 및 터널
3	팀원	이한정	국민대학교	영상처리
4	팀원	전재신	국민대학교	영상처리
5	팀원	현혜림	국민대학교	터널 및 통합

		7월		8월				9월			
		3주	4주	1주	2주	3주	4주	1주	2주	3주	4주
트랙 제작		전원									
개발환경 구축		전원									
모듈	출발 및 도로주행			현혜림	박희승 전재신	박희승 전재신					
	고가도로					박희승	박희승 김철현				
	돌발(내리막)			전재신							
	우선정지 장애물			박희승							
	곡선(S)자						박희승	박희승			
	주차(수평, 수직)			김철현	김철현	김철현					
	회전교차로				현혜림	현혜림					
	터널						현혜림	현혜림 김철현			
	차로 추월					이한정	이한정 전재신	전재신	박희승 전재신		
	신호등 분기점			이한정	이한정 박희승						
테스트 및 디버깅									이한정 전재신 김철현	이한정 전재신 김철현	이한정 전재신 김철현
통합									현혜림	현혜림 박희승	현혜림 박희승