WESC-17-06-개발계획서

금오시스

참가부문 : 자율주행자동차

팀　　명 : 금호우

팀구성원

No.	구분	성명	소속명	No.	구분	성명	소속명
1	**팀장**	**정현석**	**금오공과대학교(3학년)**	6	팀원	임석호	금오공과대학교(3학년)
2	팀원	구성현	금오공과대학교(3학년)	7	팀원		
3	팀원	권병윤	금오공과대학교(3학년)	8	팀원		
4	팀원	김상윤	금오공과대학교(3학년)	9	팀원		
5	팀원	임한솔	금오공과대학교(3학년)	10	팀원		

2017. 6.

1.

1.1. 작품 개요

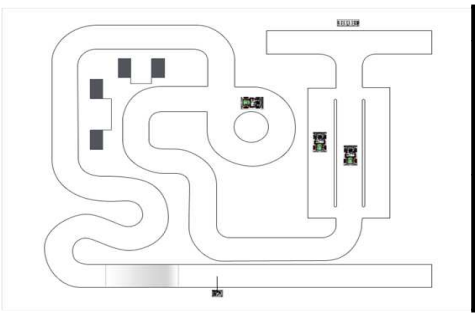
자율주행 모형자동차를 이용해서 정확하고 신속하게 주어진 자율 주행 미션을 수행하는 소프트웨어를 개발한다. 주어진 하드웨어는 카메라 및 적외선 센서를 이용해 주변 상황을 감지할 수 있고, ARM Cortex-A9 기반의 임베디드 보드에서 연산 및 판단, 지시를 내려 모터를 제어한다. 소프트웨어는 이를 위해 각종 하드웨어 제어 및 영상 처리 능력을 제공해야 한다.

본 작품에서 개발하는 소프트웨어는 역대 임베디드 소프트웨어 경진대회의 수상작들과 Github에 공개된 새로운 알고리즘 등을 참고하여 보다 개선된 알고리즘을 구현한다. 특히 입력 영상을 위에서 아래로 보는 영상으로 변환하는 버드 뷰 알고리즘을 적용하여 역대 작품들과 차별화한다. 이 알고리즘을 활용하면 모형자동차가 주행하는 선을 찾는데 있어서 지금까지 사용한 전방영상 이용 방식보다 성능을 높일 수 있을 것으로 예상한다.

또한 많은 시험 주행을 통해 각 변수들을 다양하게 적용하여 제공 받은 하드웨어와 미션 수행에 최적화된 소프트웨어 개발을 목표로 한다. 나아가 미래에 실제 도로에서 사용될 수 있는 자율주행 자동차 개발에 참여할 수 있는 경험과 지식을 축적할 것이다.

1.2. 목적

자율주행 모형자동차의 첫 번째 목표는 경기 규정에 명시된 코스(그림1)를 완주하는 것이다. 미션은 일반적인 차선 주행 외에 자동 주차(수직, 수평), 우선정지 장애물 통과, 차선 변경, 회전 교차로, 곡선 주행, 언덕 주행, 신호등에 의한 갈림길 선택 주행, 언덕 주행 등 12가지로 구성된다. 각각의 경우를 감지하고 판별하여 적절히 반응할 수 있는 소프트웨어를 개발하며, 특히 미션의 순서 변경이나 조도 등의 환경 변화에도 대응할 수 있도록 한다. 특히 표1에 나타나 있는 실격 요소는 철저히 회피하는 방식으로, 감점 요소는 최소화하는 방향으로 프로그램을 설계하고 구현한다.



1

구분	내용	
실격	완전 차선 이탈	
	장애물을 충돌하여 주행이 어려운 경우	
	3분 이내 코스 주행 실패	
	미션 중 한 개라도 정상적인 인지 불가	
감점 (회당 –5)	일시적 차선 이탈	신호등 구간 정지라인 정지X
	장애물 충돌	신호등 구간 정지 후 부저X
	주차 후 부저 X	최종 정지라인에서 정지X

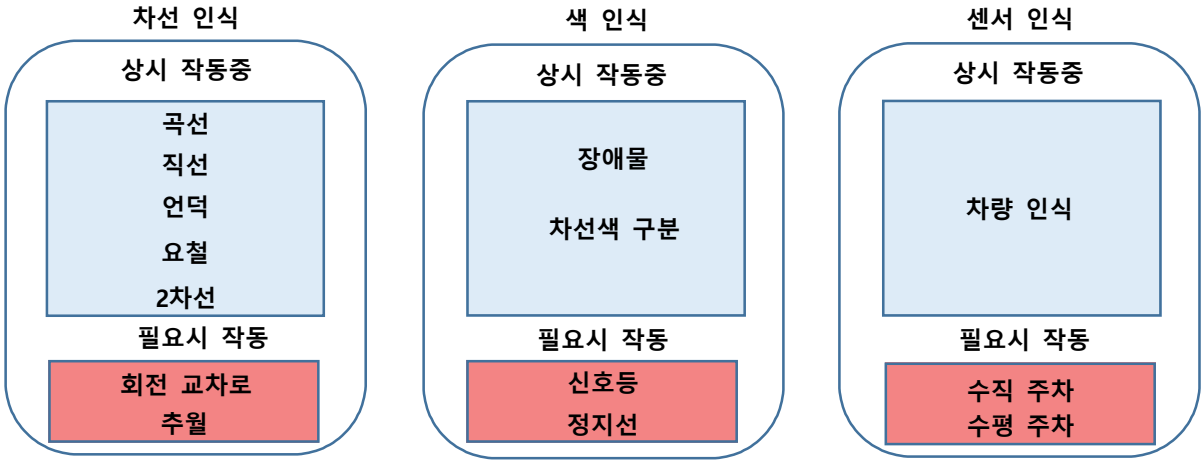
1

자율주행 모형자동차의 두 번째 목표는 최대한 신속하게 주행하는 것이다. 가장 시간이 많이 걸릴 것으로 예상되는 영상처리 부분을 최적화하기 위해 다양한 알고리즘을 시험하고 최적화하며, 센서 입력 처리와 모터 제어 등에서 하드웨어 특성을 보다 잘 반영할 수 있는 방법을 모색한다.

2.

2.1 결과물의 구성

프로젝트의 최종 결과물은 주최측에서 제공하는 하드웨어와 그를 활용하는 소프트웨어로 구성된다. 주요 하드웨어 구성 요소는 임베디드 보드, 카메라, 적외선 센서, 모터 등을 포함하고, 소프트웨어는 리눅스 기반의 운영체제와 제시된 미션을 수행할 수 있게 구성된 각종 모듈로 이루어진다. 전체 시스템 구조도는 2와 같다.



2

2.2 결과물 사양

1) 하드웨어 구성

NO.	항목	규격 및 내용	비고
1	크기	330 * 190 * 160mm	
2	무게	약 2300g	
3	최고속도	1000 $mm/\sec$	
4	구동 모터	DC서보 모터 엔코더(12V-12W)	하프브릿지
5	임베디드 보드	NVIDIA Tegra3 (ARM Cortex-A9 Quadcore 900MHz)	DDR 2GB NOR Flash 64MB eMMC 8GB Ti로 교체
6	카메라	CMOS VGA급(640*480), CVBS 출력	
7	디버깅	Ethernet(10/100) 및 mini USB	
8	서브 모터 – 3EA (바퀴 조향 및 카메라 틸트)	6V, 9kg(180°)	
9	거리센서	적외선 거리센서 – 6SET (4cm~ 30cm)	
10	적외선센서	라인감지용 7조	오토 튜닝
11	전압 / 배터리	NIMH 12V - 3000mA	방전율15C 하프브릿지
12	통신	SPI, RS232C: 19200BPS(고정)	
13	충전 및 작동 시간	약 90분 충전, 약 2시간 30분 사용	
14	전조등, 방향등, 정지등		

2

2는 대회 개요에 명시된 자율주행 자동차 장비 사양이다. 이 부분은 소프트웨어 개발에 중요한 요소로 하드웨어 입력 값에 따른 한계 요소를 개발 단계에서 고려해야 한다.

2) 소프트웨어 구성

분류	모듈	기능
주행	직선차로	BirdView기법과 Otsu's method를 이용한 차선 검출
	곡선차로	Sliding Window기법을 통한 곡선 차로 검출
	요철구간	노이즈 제거를 위한 Closing 기법을 이용
	회전교차로	영상처리와 적외선센서를 통한 장애물 인식
주차	주차공간인식	적외선센서를 통한 주차공간의 폭과 너비 인식
	수평/수직주차	각 주차공간에 해당하는 주차 알고리즘 실행
장애물	돌발구간	이진화한 영상을 통한 표지판 인식
	언덕구간	Hough Line Transform 알고리즘을 이용한 소실점 계산과 언덕 인식
	차로추월구간	장애물 인식 시 양쪽 Window를 그려서 확인하는 알고리즘
	신호등	Hough Circle Algorithm을 이용한 신호등 인식

3

3 개발할 소프트웨어의 전체적인 구성으로 총 12가지 미션(수평/수직 주차, 직선차로와 2차로 주행은 하나로 표현)을 처리하기 위한 기능이다. 이 알고리즘들을 쓰레드에 어떻게 분배해야 할지를 정해야 하지만 현재 GPU 코어의 사용 가능 여부를 모르기 때문에 분배를 결정하기 힘들다. 개발 단계에서 코어마다 쓰레드 분배 결과에 따른 성능을 비교 해보고 최고의 성능을 발휘하는 분배법을 찾아낸다.

3.

3.1 기술적 요구사항

1) 하드웨어 분석

표2는 자율주행 자동차 장비 사양이다. 소프트웨어를 개발하기 전이나 개발하는 도중의 모든 요소를 고려해야 하지만 특히 주요 요소로 모터(구동 모터, 서보 모터), 카메라(서보 모터를 통한 각도 조절), 적외선 센서, 전압 및 배터리 충전이 있다.

- **모터 제어**

모터는 언덕 구간과 요철 구간에서 동력이 부족할 경우 제대로 주행을 못 할 수 있다. 그러므로 구간을 인식하여 동력을 적합하게 조정해야 하며 적정값은 테스트를 통해 설정한다. 또한, 전체적인 주행에서 속도가 지나치게 빠르면 영상의 전환속도가 지나치게 빨라져서 처리에 문제가 생기기 때문에 적합한 속도를 역시 테스트를 통해 찾는다.

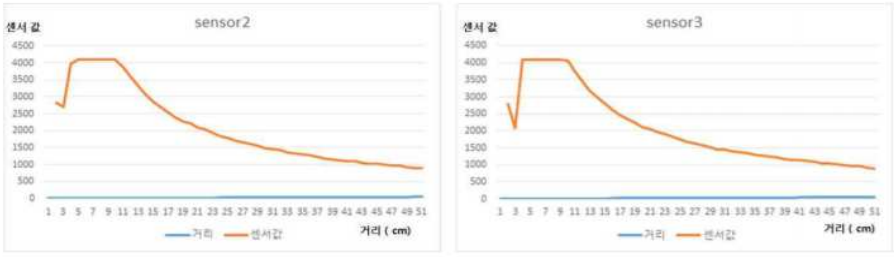
- **카메라 각도**

영상처리 작업을 위해 적합한 영상 입력을 받기 위해서 카메라 각도를 조절해야 한다. 적합한 영상

이란 차선, 장애물, 신호등 등을 구분하고 처리하는 알고리즘을 수행하는데 상황마다 필요한 요소들이 입력된 영상이다. 2016년 수상작(Kudos-V)의 결과보고서를 참고해 보면 정면에서 14도 정도 내린 각으로 ROI(Region of Interest)를 잘라서 처리하였다. 다른 팀들도 대개 자신에게 적합한 각을 찾아서 사용했으며, 우리도 테스트를 통해서 적정한 카메라 각도를 찾을 계획이다.

- **적외선 센서**

적외선 센서에서 입력되는 값에는 신뢰 구간이 있음을 고려한다. **그림3**은 2014년 최우수 팀(인카)의 결과보고서에 나온 감지 값이다. 이를 참조하면 10cm 미만에서는 유의미한 값이 나오지 않음을 알 수 있다. 그러므로 주행 시 한쪽 차선에 치우쳐 주행하면 적외선 센서 입력 값이 너무 가깝거나 멀기 때문에 값을 분석하기 힘들어진다. 따라서 프로그램 설계 및 개발 단계에서 주행 시 한쪽 차선에 치우치지 않는 알고리즘을 적용하여 이 문제점이 해결되도록 한다.



3 2014 ( )

- **배터리**

자율주행자동차는 배터리로 동작하며, 충분한 용량만큼 충전이 되어 있지 않으면 전압이 부족하여 하드웨어들이 성능을 발휘하지 못할 수 있다. 배터리를 상시로 충전하며, 가능하면 용량 부족을 체크하여 알려 주는 기능의 추가를 고려한다.

2) 임베디드 보드 최적 활용

NO.	하드웨어	내용	비고
1	프로세서	NVIDIA Tegra3 (ARM Cortex-A9 Quadcore 900MHz)	
2	메모리	DDR3 2GB	
		NOR Flash 64MB	
		eMMC 8GB	
3	카메라	CMOS VGA급(640x480), CVBS 출력	
4	USB	USB2.0 port 2EA	
5	AD	AD port 6EA (PSD sensor용)	
6	LCD	4.3" (해상도 480x272)	
7	디버깅	Ethrenet(10/100) 및 mini USB	
8	보드간 통신	SPI 및 UART	

4 (NVIDIA Tegra3)

NVIDIA Tegra3 임베디드 보드는 core가 4개로 총 4개의 CPU를 사용하는 프로그램을 구성할 수 있다. 즉 CPU 쓰레드를 총 4개 사용할 수 있다. 설계 단계에서 이 쓰레드들을 어떻게 구성하여 사용할 것인지 계획되어야 한다.

NVIDIA Tegra3 processor(http://www.nvidia.com/object/tegra-3-processor.html)의 사양을 보면 GPU core가 12개 있는 것을 확인할 수 있다. 이 core들은 GPGPU 기술을 활용해 영상 처리의 속도를 높이는데 큰 역할을 할 수 있다. 따라서 GPGPU 기반으로 구현된 OpenGL, OpenCL 등의 라이브러리를 활

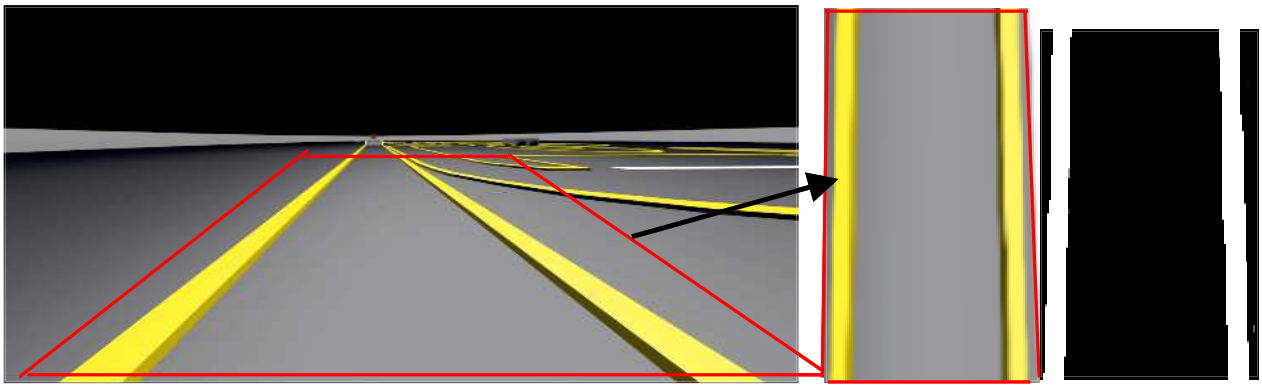
용하는 방법을 고려하여 고속 주행의 목표를 달성할 수 있도록 한다.

3) 직선 차로 주행

- 버드 뷰 알고리즘(Bird View Algorithm)

버드 뷰 알고리즘은 3D 영상을 위에서 아래로 보는 이미지로 변환시키는 기술이다. 이 기술을 이용하면 차로 주행 뿐 아니라 이외 영상을 처리하는 것도 아주 간단해진다. 이 알고리즘은 Udacity라고 하는 MOOCs(Massive open online courses) 강의 중 Self-Driving Car 과목에 소개된 것을 참고하였으며 소스는 Github에 제공되어 있다.

그림 4는 버드 뷰 알고리즘을 테스트해 보기 위해 직접 그래픽 작업으로 대회 경기장 규격의 가상 영상을 만들어서 영상처리를 한 것이다.



4

먼저 입력 이미지를 이용해 변환 행렬을 구한다. 이를 homography(1)라 하는데 전방 영상을 공중에서 수직으로 내려 보는 각도의 영상이 되도록 4쌍의 좌표값을 지정해 행렬의 각 요소값을 구한다. 이후 입력되는 영상은 homography를 통해 변환하여 직사각형 형태로 만들어 처리할 수 있다. **그림 4**의 두 번째는 변환한 영상이고, 세 번째는 이를 다시 그레이스케일 영상으로 변환 후 Otsu's 알고리즘을 이용하여 threshold 값을 찾아서 이진영상을 만든 것이다.

$$w\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}h_{11}\,h_{12}\,h_{13}\\h_{21}\,h_{22}\,h_{23}\\h_{31}\,h_{32}\,h_{33}\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

1 homography

- 차선 감지

그림 4의 가장 오른쪽 이진영상에서 x축을 기준으로 히스토그램을 그려보면 차선의 중간값을 찾을 수 있다. 낙타 혹 모양처럼 양쪽으로 히스토그램이 많이 검출되는 지점을 찾을 수 있다. 이 지점이 직선구간에서 양 차선이므로 양쪽에서 뭉치는 값의 평균값이 구동체가 주행할 차선이다.

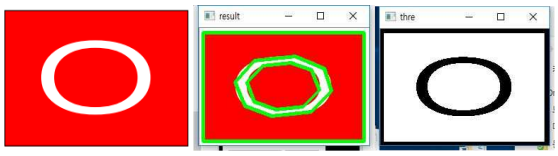
이를 이용하여 현재 자동차가 정상적인 주행을 하고 있는지에 따라 구동체에 시리얼 통신을 통하여 신호를 주어서 위치를 찾아 갈 수 있다.

- 요철 구간

요철 구간은 직선 차로의 확장이다. 하지만 현재 그레이스케일로 변환하여 처리하는 경우에 요철이 차선으로 인식될 수 있다. 말하자면 요철은 차로 주행에 있어서 노이즈가 발생하는 것이다. 이 노이즈를 제거하는 알고리즘으로 Closing 기법이 있다. Closing 기법은 형태가 잡혀있는 인식된 물체의 테두리를 깎고 다시 채우는 알고리즘인데 이를 이용하면 영상의 노이즈를 제거할 수 있다. 요철 구간에서

도 노이즈에 해당하기 때문에 Closing 기법 사용 시 테스트를 통해 차선으로 검출되지 않도록 마스크 값을 지정하여 없애도록 한다.

4) 돌발 상황 대처



5

돌발 상황 대처는 **그림5**에서 맨 좌측에 보이는 표지판을 인식했을 때 멈춘다. 이 표지판은 빨간색, 큰 사각형, 안에 있는 원으로 특징지을 수 있다.

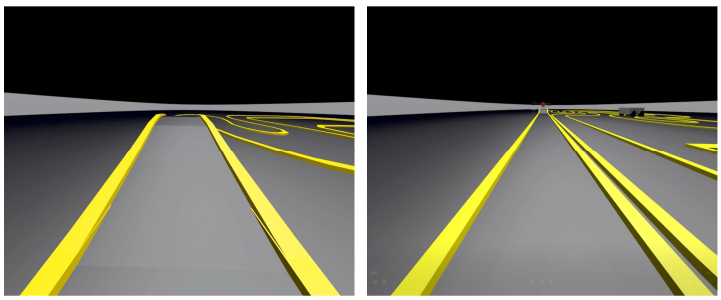
이 특징들을 검출하기 위해서 먼저 RGB를 HSV로 변환하면 조명의 변화에도 색상을 구분할 수 있게 된다. Hue를 빨간색으로 지정하여 이를 기준으로 영상을 이진화 한 후 사각형과 원을 인식하도록 한다. 사각형 검출은 Douglas-Peucker 알고리즘을 이용하여 곡선을 제거하고 직선인 선을 찾아서 각이 90°인 꼭짓점 4개를 찾으면 된다. Hough Circle Algorithm을 통해 원을 인식하면 시리얼 통신을 통해 구동체에 정지 신호를 보낸다.

그런데 역대 수상작을 보면 굳이 원을 인식하지 않고 빨간색 사각형만으로도 표지판 인식이 가능하다는 결론이 도출된 팀도 있었다. 또한, 이 방법 외에도 Neural Network를 이용하는 방법이 있는데 이 방법과 성능을 비교해 보고 좋은 방법을 사용할 예정이다. 단, Neural Network를 이용하는 방법은 높은 시스템 사양을 요구하기 때문에 실현 가능성이 적다고 판단된다.

```
Begin
  1) 이미지를 HSV로 변환
  2) 빨간색이 나타내는 공간을 지정. 및 이진화(Threshold)
  3) 이진화 된 이미지의 외곽선 검출
  4) 검출된 외곽선을 DP알고리즘을 이용, 다각형으로 근사화
  5) 근사화된 윤곽의 개수가 4개이고 예각인 부분을 선으로 지정 및 검출
end
```

1.

5) 언덕 인식



6

직선 찾기 알고리즘을 이용하여 차로를 구하게 되면 각 차선의 직선을 구할 수 있다. 주행 중 차선의 기울기 값을 측정하여 언덕인지 판단할 수 있다.

언덕 차선은 직선 차선과 차선 기울기가 달라진다. 언덕 주행의 경우 양쪽 차선이 직선차선에 비해 절대 값 기준으로 각도가 커진다. 이 지점을 테스트를 통해 언덕 차로 각도를 측정하고 직선 차로 각도를 측정하여 구동체가 모터를 더 강하게 돌리도록 신호를 보내면 된다.

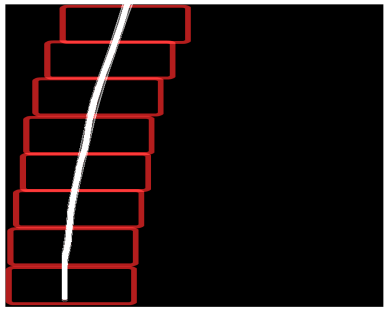
직선 찾기 알고리즘은 흔히 사용하는 Hough Line Algorithm과 RANSAC알고리즘 중 테스트 결과를 통해 성능이 좋은 알고리즘을 이용할 계획이다. 흔히 Hough Line Algorithm은 선을 찾는 성능은 좋으나 노이즈에 약하고 찾는 속도가 매우 느리다. 반면, RANSAC 알고리즘은 무작위로 점들을 찾아 연결

하기 때문에 정확도는 떨어지나 선을 찾는 속도가 빠르고 노이즈에 강하다. 그러므로 RANSAC을 우선적으로 사용하여 차선을 정확히 추출하는지 확인하고 사용에 무리가 있다고 판단되면 Hough Line Algorithm을 통해 차선을 찾을 계획이다.

6) 곡선 차로 주행

- Sliding Window

차선을 검출하기 위해서 Sliding Window란 기법을 사용한다. Sliding Window란 하나의 차선을 여러 개의 Window로 나누어 검출하는 방법이다. 각 Window의 탐색범위를 이미지로 표현하면 **그림 7**과 같은 형태로 나타난다.



7 Sliding Window

Sliding Window에서 성능을 결정짓는 값으로 Window 개수와 최소픽셀 수가 있다. Window 개수는 어느 정도로 분할하여 검출할지를 결정하고, 최소픽셀 수는 어느 정도 픽셀까지 차선이라 판단할지를 결정한다. 두 변수의 값에 따라 성능이 달라지기 때문에 여러 번의 테스트를 통해 가장 최선인 값을 결정하여 사용하여야 한다.

검출된 차선에서 하나의 최적선을 도출하는 기법으로는 최소자승법을 이용한다.

```
input  : minPixelCount, xOffset ( histogram을 통해 구한 첫번째 윈도우의 중앙값 ),
         xMargin ( 탐색 범위 ), image
output : lane ( 차선에 포함된 모든 픽셀좌표 )
lane ← []
while count ←1 to window개수
  Pixels ←[]
  nowPixelCount ← 0
  nowY ← image.height - ( image.height/window개수 * count )
  for nowX ←xOffset - xMargin  to  xOffset + xMargin
    if image[nowX][nowY] 에 픽셀이 존재 then nowPixelCount ← nowPixelCount + 1
      Pixels add ( nowX, nowY )
  if minPixelCount < nowPixelCount then xOffset ←Pixels의 x값들의 평균값
    lane add ( Pixels )
```

2. Sliding Window

$$
\begin{aligned}
(n)a_0 + (\sum x_i)a_1 + (\sum x_i^2)a_2 &= \sum y_i \\
(\sum x_i)a_0 + (\sum x_i^2)a_1 + (\sum x_i^3)a_2 &= \sum x_i y_i \\
(\sum x_i^2)a_0 + (\sum x_i^3)a_1 + (\sum x_i^4)a_2 &= \sum x_i^2 y_i
\end{aligned}
$$

2

※ 최소 자승법 : 차선검출로 나온 모든 좌표 값들을 2의 공식에 대입하여 a_0, a_1, a_2 값을 구한 뒤 $y = a_2x^2 + a_1x^1 + a_0$에 대입하여 최적방정식을 구한다.

- 곡률 반경

곡률 반경을 통하여 커브의 정도를 파악할 수 있다. 곡률반경이란 어떠한 좌표들을 원의 바깥부분이라고 가정하고 이들을 지난 최적의 원이 가지는 중심점과의 거리를 의미한다. 값이 커질수록 방정식은 직전에 가깝고 멀수록 곡선에 가깝다. **수식3**를 이용하면 곡률 반경을 구할 수 있다. 양 차선의 방정식을 더하고 2로 나눠서 중앙 차선의 방정식을 구한 후

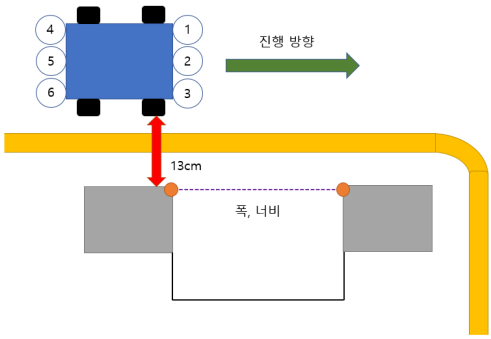
$$
R = \frac{(1 + (\frac{dy}{dx})^2)^{\frac{3}{2}}}{\left| \frac{d^2y}{dx^2} \right|}
$$

3

중앙 차선의 곡률반경을 구하여 곡선 차로인지 판별하고 구동체를 조작하도록 한다.

7) 주차

대회 규정상 주차 공간은 직선도로 상의 임의의 위치에 놓이며, 주행 차선 라인(바깥쪽)과 약 13cm의 간격을 두고 설치된다. 그리고 주차 공간 양 옆에 직육면체의 장애물이 위치해 있다. **그림8**을 참고하면 6개의 거리센서 중 6번 센서를 이용해 주차공간 감지를 수행한다. 직육면체의 끝과 끝을 감지해서 주차 공간의 폭을 인식하고, 폭이 규격에 일치하면 폭에 따라 수평 또는 수직 주차 알고리즘을 실행한다.

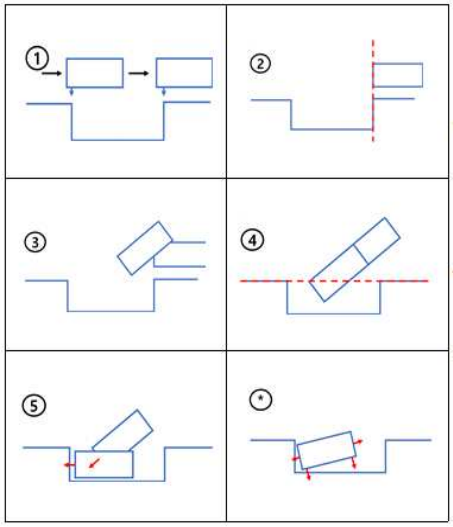


8

- **수평 주차 알고리즘 (그림9 참고)**

1) 주행하며 후방 센서(6번)로 주차 공간(폭과 너비)을 측정
2) 공간 끝을 발견하면 차량의 후방 선을 주차라인까지 이동 후 정지
3) 조향을 오른쪽으로 최대로 틀고 후진
4) 뒷바퀴 주차라인까지 후진
5) 조향을 왼쪽으로 최대로 틀고 센서(5번) 측정을 하며 후진
※ 만약, 주차가 불완전하게 될 시
벽면의 두 개의 센서(3번, 6번)가 바라보는 거리가 같아질 때까지,
전후방 센서(2번, 5번)의 거리가 같아질 때까지 주차 마무리 알고리즘 반복

3.

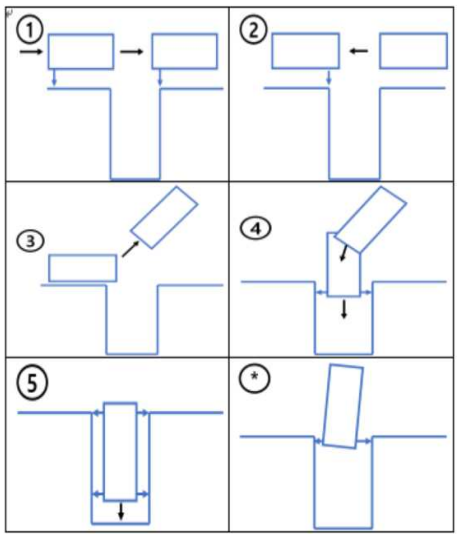


9

- **수직 주차 알고리즘 (그림10 참고)**

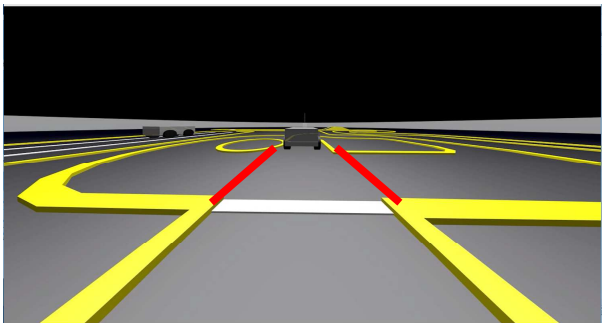
1) 주행하며 후방 센서(6번)로 주차 공간(폭과 너비)을 측정을 측정하고 공간 끝을 발견하면 정지
2) 후진하며 전방 센서가 주차 공간을 인식하면 정지
3) 조향을 왼쪽으로 최대로 틀고 전진
4) 우향 후진하며 후방 양쪽 센서(4번 6번)에서 값이 추출되면 벽과 부딪히지 않을 적정 거리를 유지하며 후진
5) 전방 센서(1번, 3번)에서 값이 추출되면, 4개의 센서(1번, 3번, 4번, 6번) 값을 비교하며 후진
※ 주차 진입각 재시도 알고리즘
후진하며 주차 공간 진입시 한쪽 벽면으로 너무 치우친 경우 후방 양쪽 센서(4번, 6번)의 값이 같을 때까지 재진입

4.

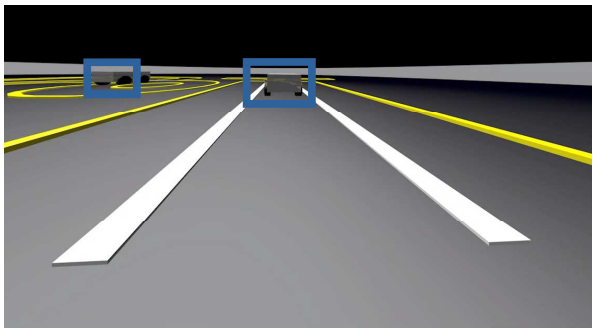


10

8) 회전 교차로



a.



b.

11

회전 교차로는 노란 차선이 끊어지고 수평 방향으로 하얀 정지선이 나타나는 곳이다. 끊어진 노란 차선을 이어서 보면 일반적인 곡선 주행으로 처리할 수 있다. 그러나 회전교차로에서는 장애물(선행 차량)이 있을 때를 처리해야 한다. 처음 진입할 때 우선 정지선에 멈춘 후 좌측 교차로에서 차가 나오지 않으면 진입한다. 그리고 주행을 하다가 장애물을 감지하면 속도를 늦추어 앞 차와의 거리를 유지하도록 한다. 이 때는 **그림8**의 2번 센서와 영상처리 과정을 통해 판단을 하며, 장애물이 사라지면 속도를 복구한다.

이러한 간단한 알고리즘을 통해서 쉽게 회전 교차로를 통과 할 수 있다고 보지만 인식 과정에서 문제가 생길 수 있으므로 개발을 하면서 테스트를 통해 좋은 아이디어를 선택해서 적용해야 한다.

9) 차로 추월 구간

일반 차로와 차선 차로를 구분할 수 있는 특징은 선이 끊어지는가, 색이 어떤 색인가에 따라서 구분할 수 있다. 일반 차로의 경우에는 노란색 차선과 실선이기 때문에 별 무리 없이 직선 차로에 진행하는 것처럼 인식하면 된다. 그러나 차선 차로를 구분하려면 흰색을 기준으로 영상을 이진화해서 끊어지는가를 판단하면 된다. 알고리즘을 더 단순화하면 흰색 차선만 구분하면 된다.

그리고 앞에 장애물이 두 개가 나타나기 때문에 진입하면서 어느 차로로 이동할지 판단해야 한다. 우선 전방에 장애물을 인식한 후 왼쪽 오른쪽에 Window를 하나씩 그린 후 좌측과 우측에 장애물이 있는지 확인한다. 이후 판단에 따라 차선을 변경할 수 있도록 구동체에 신호를 전달한다.

10) 신호등

정지선에서 일단 정지 후 신호등 ROI 영상으로부터 신호등이 검출되면 이를 그레이스케일 영상으로 변환하고 Hough Circle Algorithm을 이용해 4개 원의 중심점을 찾는다. 주기적으로 각 원의 중심점 부근의 색을 검출해 색상을 인식하고 변화를 감지한다. 세 번째 원에서 변화가 있을 경우 좌회전하고 네 번째 원에 변화가 있을 경우 우회전한다.

문제는 각 원의 반지름 값을 모를 경우, Hough Circle Algorithm은 모든 픽셀에서 원을 그리면서 해당 원이 지나는 곳에 flag값을 증가시키면서 원을 찾기 때문에 시간이 오래 걸린다는 단점이 있다. 그러므로 테스트를 통해 반지름에 근사한 값을 알아 내어 알고리즘 수행 속도를 향상시킬 예정이다.

그레이스케일 영상을 사용하지 않고 각 색상의 threshold 값을 정해서 신호를 검출하는 알고리즘도 존재하지만 이는 매우 느린 것으로 알려져 있다. 그러나 정확도 측면에서는 더 나을 수도 있기 때문

에 각 알고리즘을 모두 구현하고 테스트를 통해 정확하고 신속하게 판별할 수 있는 최적의 알고리즘을 설계하고자 한다.

11) 최종 정지

신호등에 따라 우회전 또는 좌회전을 한 후 안전 지대를 감지하여 경계선 내에 최종 정지해야 한다. 안전지대는 **그림12**와 같이 노란색과 흰색의 빗금으로 표시된다. 그 이전에 하얀 정지선을 인식하여 일단 정지하고 영상을 분석하여 노란색 흰색이 엇갈려 나타나는 것으로 인식되는 경우 정지를 준비한다. 적정 진행 거리와 정지 지점은 실험을 통해 정할 예정이다.

12

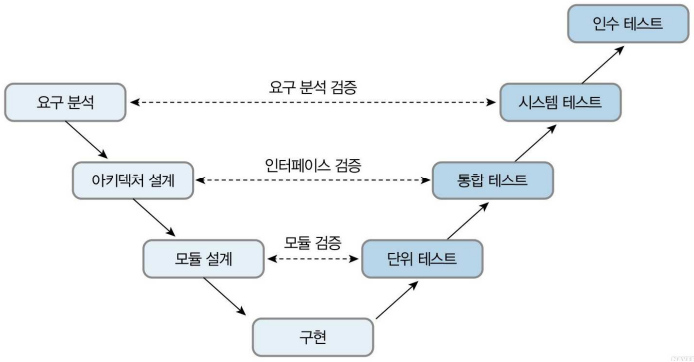
3.2 개발방법

1) V 모델

소프트웨어 개발 프로세스 중, 폭포수 모델의 확장된 형태인 V모델을 적용하여 개발할 것이다.

V모델은 소프트웨어 개발의 각 단계마다 상세한 문서화를 통해 작업을 진행하는 잘 짜인 방법을 사용한다.

테스트 설계와 같은 테스트 활동을 코딩 이후가 아닌 프로젝트 시작 시에 함께 시작하여, 전체적으로 많은 양의 프로젝트 비용과 시간을 감소시켜 현재 프로젝트에 적합한 방법론이라고 생각된다.



13 V

현재 팀원들의 역할이 분배되어 있는데 코드마다 전부 모듈화하여 모듈마다 테스트를 진행하여 오류 및 구동상태를 확인할 것이다. 그리고 이 모듈마다 베타 테스트를 진행하며 오류를 수정한다.

2) GIT & GitHub

Git은 소스의 버전관리를 위한 Tool로 개발 기록들을 관리할 수 있다. 앞서 이야기한 V 모델을 이용하면 해당 프로젝트는 개발 과정에서 버전이 업데이트될 수밖에 없다. 그러므로 날짜와 각 단계마다 어디까지 진행되었는지 알 수 있는 정보가 필요할 때 유용하다.

GitHub는 Git을 사용하는 프로젝트를 지원하는 웹호스팅 서비스로 소스 관리를 지원한다. Git에서 사용하는 프로젝트에서 팀워크를 하는 도중 변수나 함수 이름 전체적으로 바뀌었을 경우 난해할 수밖에 없다. 그렇기 때문에 소스를 관리해주는 툴이 필요한데 GitHub는 프로젝트 관리를 위한 Wiki, 버전 관리, 기록 등 다양한 기능들을 제공한다. 그 외에도 팀원들이 각기 다른 부분을 수정하여도 이를 합치기가 수월하게 되어있어서 개발하는 데 있어서 유용하다.

현재 프로젝트에서는 서로 간의 협업을 위해서 GIT과 GitHub를 사용할 예정이다. 물론, 유료로 사용하지 않으면 강제로 소스를 오픈해야 하는 단점이 있지만 상업적인 목적이 아니므로 상관없다. 문제가 생길 경우 유료로 전환하여 사용할 예정이다.

3)모듈화 및 객체화

프로젝트의 개발 과정에서 오류가 발생할 수 있고 이 오류를 수정할 때 전체적인 프로그램이 변경되면 안 된다. 그리고 협업하는 과정에서 서로 겹치는 부분을 따로 개발하면 낭비가 될 수 있다. 이 문제점을 해결하기 위한 방법이 모듈화, 객체화이다.

모듈화는 프로그램을 모듈 단위로 나누는 것이다. 팀원들끼리 역할을 분배하여 서로 다른 모듈을 개발한다. 그리고 이 모듈들을 한 번에 합쳐서 프로그램을 돌려보고 만약에 이상이 생기면 디버그를 통해 어떤 모듈에서 오류가 발생했는지 확인한다.

대부분 소프트웨어는 여러 번의 시행착오를 통해 완성된다. 예상치 못한 예외가 발생할 수도 있으므로 팀원들 간에 프로젝트에 관한 회의를 통해 이러한 문제점을 찾고 수정을 해 나가면서 조금씩 프로젝트 크기를 키워나가면 된다.

4) 개발 도구

- **운영체제 및 라이브러리**

임베디드 소프트웨어 경진대회 홈페이지에 있는 Q&A 확인 결과 이전 참가자들이 Ubuntu 12.04 LTS OS 개발환경에서 작업했다는 것을 알 수 있었다. 이 외에 SDK 및 Tegra전용 OpenCV 정보는 대회주최 측에서 자료를 제공해 주기 때문에 큰 문제 없이 설치가 될 수 있을 것이라고 생각된다.

그러나 대부분의 수상작 결과보고서에 OpenCV를 사용하면 성능이 많이 저하된다고 언급되어 있다. 그러므로 OpenCV를 설치해서 테스트를 해보고 만약 성능이 아주 느리다면 직접 라이브러리를 구현하여 사용할 예정이다. 만약 GPU 기반의 OpenCL, OpenGL이 지원되어 사용가능하다면 해당 라이브러리를 가지고 와서 개발에 사용할 것이다. 가능하다면 하드웨어의 모든 기능을 활용하여 프로그램의 성능을 개선할 계획이다.

- **gcc/g++**

C/C++이 주요 개발 언어이므로 gcc/g++ 컴파일러를 사용한다. C/C++은 원하는 하드웨어의 기능을 사용하기 적합하고 팀원들에게 가장 익숙한 언어이기 때문에 선택하였다.

C/C++로 구현된 OpenGL, OpenCV, OpenCL, OpenMP 등 사용할 수 있는 라이브러리가 많으므로 이를 활용하여 개발하면 성능을 높일 수 있을 것으로 예상한다. 그러나 라이브러리 사용에 따라 성능이 저하되는 경우도 있기 때문에 테스트를 해보고 최적의 결과가 나오는 라이브러리를 선택하여 사용할 예정이다.

4.

1) RANSAC 반복 횟수

언덕 인식 과정에서 차선 후보들을 이용하여 RANSAC 알고리즘으로 소실점을 구한다. RANSAC알고리즘은 샘플을 반복적으로 선택하여 구하는 방식인데, 반복횟수가 많을수록 더 정교한 소실점이 구해지게 된다. 그러나 언덕 인식 뿐 아니라 다양한 쓰레드들이 병행하여 수행되고 있으므로 RANSAC 알고리즘으로 인해 전체적으로 느려질 가능성이 있다. 따라서 언덕을 인식할 수 있게 하고 반복 횟수는 최소화한다.

2) Tegra3 보드에서 OpenCV 사용 시 발생하는 문제

OpenCV는 컴퓨터 비전 라이브러리로 다양한 버전으로 배포되어 사용되고 있으며 배포되는 언어나 운영체제에 따라서 발휘하는 성능은 각기 다르다. Tegra3 보드에서는 대회 주최 측에서 제공하는 SDK 파일을 이용하여 설치하는데, 역대 참가팀을 보면 OpenCV 사용 시 프로그램 성능이 저하되기 때문에 직접 영상처리 알고리즘을 구현하여 사용한 팀이 대다수이다. 그러므로 개발과정에서 중간 테스트 과정을 통해 성능을 평가하고 OpenCV 라이브러리 사용 시 주행에 문제가 생기면 직접 라이브러리 구현을 통해 해결한다.

3) 조명에 따른 영상처리 문제

RGB 영상을 처리하면 조명 반사 문제 때문에 정확한 임계값을 정하기가 힘들다. 그러므로 HSI값으로 변환하여 임계값을 정하기 쉽도록 한다. RGB는 이름처럼 색으로 구분하기 편한 좌표계인 반면, HSI는 색보다는 밝기와 채도에 민감한 좌표계이므로 조명으로 인해 반사된 색을 구분하기에 더 적합하기 때문이다. 수식5는 RGB 좌표계 영상을 HSI 좌표계로 바꾸는 공식이다.

$$I = \frac{1}{3}(R+G+B)$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]$$

$$\text{if}(B \leqq G)then$$

$$H = \cos^{-1}\left(\frac{\frac{1}{2}[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

$$else, H = 360 - H$$

5 RGB -> HSI

각 임계값은 테스트를 통해 결정하는 것이 옳다. 왜냐하면 카메라에 따라서도 입력되는 영상 값이 전부 다르게 나오기 때문에 테스트를 통해 최적인 값을 찾아서 영상을 처리해야 한다.

4) 거리센서 사용에서의 제한

거리센서는 3cm ~ 40cm가량의 거리를 정밀하게 측정하는데 사용된다. 그러나 2014년 최우수팀(인카)의 결과보고서에서 나온 감지 값을 확인해보면 10cm 미만에서는 유의미한 값이 나오지 않는다. 위와 같은 방식으로 거리센서를 사용하였을 때 주차를 하다가 후방 센서의 값을 통해 정지를 하게 되면 코딩에서 생각했던 것 보다 훨씬 전에 조기 수렴하여 그 값을 인식한 상태로 멈추게 되는 현상이 발생할 수 있다.

해결방법으로 2012년도 최우수 팀의 결과보고서를 참고하면 위와 같은 문제를 라이브러리 상에서 받아오는 출력 값을 char형에서 short형으로 변환함으로써 범위를 더 세세하게 나누어서 40cm부터 3cm값까지 동일한 값이 분포하지 않도록 결과 값을 바꾸어 준다. 이렇게 되었을 때 주차코스에서 거리센서를 이용한 값을 활용함에 있어서 보다 정확도가 높아지는 결과를 얻을 수 있다.

5) 쓰레드를 분배하는 방법

쓰레드를 할당하는 문제는 프로그램 성능과 관련된 가장 큰 문제이다. 쿼드코어의 CPU를 가지고 있기 때문에 4개의 쓰레드를 분배할 수 있지만 GPU Core를 사용할 수 있으면 훨씬 더 성능을 개선할 수 있다. 그리고 정해진 쓰레드 수 내에서도 어떻게 분배를 할지 여러 가지 방안이 가능하다. 제공되는 임베디드 보드를 사용해 보지 않았기 때문에 상세한 구성은 추후 결정할 예정이다.

5.

분류		구분	7월				8월				9월			
			1	2	3	4	1	2	3	4	1	2	3	4
개발 환경 구축		테스트용 경기장 제작												
		테스트용 영상 제작												
코스별 모듈 개발&테스트	차선 인식	직선 구간												
		언덕 구간												
		곡선 구간												
		추월 구간												
	장애물 인식	돌발 상황												
		신호등												
		회전 교차로												
		요철 구가												
		2차선 구간												
	주차	수직 주차												
		수평 주차												
	차량 제어	속도 제어												
		방향 제어												
		위치 제어												
통합 & 테스트		모듈 종합												
		종합 테스트												
		프로그램 보완 및 최적화												
		최종 테스트												
		결과 보고서 작성												

정현석 임한솔 김상윤 공동

임석호 권병윤 구성현

6.

역할	이름		책임
총 매니저	정현석		- 프로젝트 총괄 - 팀원 간 의견 조율
개발 환경구축	공동		- 테스트용 경기장 제작 - 테스트용 영상 제작
코스별 모듈 개발 & 테스트	차선 인식	임석호	- 직선 구간 - 곡선 구간 - 추월 구간
		임한솔	- 언덕 구간
	장애물 인식	권병윤	- 회전 교차로 - 2차선 구간
		정현석	- 돌발 구간 - 신호등
		임한솔	- 요철 구간
	주차	구성현	- 수평 주차
		김상윤	- 수직 주차
	차량 제어	구성현	- 위치 제어
		김상윤	- 속도 제어
		권병윤	- 방향 제어
통합 & 테스트	공동		- 모듈 종합 - 종합 테스트 - 프로그램 보완 및 최적화 - 최종 테스트 - 결과 보고서 작성

7.

1. 제 15회 임베디드 경진대회 게시판 (역대 수상작, Q&A 등) (eswcontest.com)
2. Udacity Self-Driving Car Git자료 및 lecture (https://www.udacity.com/)
3. 영상 처리 수업 자료(금오공과대학교 한규필, 김성영 교수님)
4. 다크 프로그래머 블로그 – 영상처리 기술(RANSAC, Haar/Cascade Training 등) (http://darkpgmr.tistory.com)
5. 자율 주행 자동차 관련 YouTube 동영상(신호등 추출, 표지판 추출) - ROI Idea (https://www.youtube.com/watch?v=2-vFB-12Zm4)
6. Wikipeida(위키피디아) (Hough transform, homography 등 적용한 알고리즘 원리)
7. Tegra3 Board 사양 (http://www.nvidia.com/object/tegra-3-processor.html)
8. OpenCV Document(http://opencv.org/)
9. Bird Eye View Map 관련 논문(www.mdpi.com/1424-8220/12/4/4431/pdf)