2016 지능형모형차 경진대회 보고서

학 교	한양대학교
팀 명	우승꿍꼬또
유 형	본경기
팀 장	연규환 (미래자동차공학과)
팀 원	김대현 (미래자동차공학과) 김정훈 (미래자동차공학과) 송현섭 (미래자동차공학과) 이홍규 (미래자동차공학과)

1 개요

1.1 설계 배경

최근 자동차 업계에서 가장 떠오르고 있는 화두는 무인 자율 주행 자동차이다. 현대자동차를 비롯한 GM, 폭스바겐, BMW 등 대부분의 자동차 회사들이 무인 자율 주행 자동차를 실현시키기 위해 많은 자본과 연구 인력을 투자하고 있으며 자동차 기업이 아닌 Google, Apple 등의 IT 기업들도 무인 자동차 연구개발에 참여하고 있다. 무인자동차 개발을 위해서는 사람의 신체 기관을 대신하여 도로 상황을 인지하고 최적의 주행을 위한 판단을 내리기 위해서는 수많은 센서들에서 얻어진 Signal 들을 이용하여 MCU 를 통해 제어를 해주어야 하기 때문에 제어와 센서 분야의 전문가들이 필수적이다. 이번 2016 지능형 모형차 경진대회를 통해 센서와 MCU 를 통한 차량 제어를 직접 경험함으로써 무인 자동차 개발의 전문가가 되기 위한 경험을 쌓고자 설계하게 되었다.

1.2 설계 목표

무인 자동차에서 가장 중요한 목표는 인식, 판단, 주행이다. 현재 자동차의 상황을 정확히 인식하고 최적의 판단 과정을 거쳐 맞는 주행을 해야 무인 자율주행 자동차라고 할 수 있다. 2016 년 지능형 모형차 경진대회 본 경기에서도 인식, 판단, 주행을 통해 지정된 트랙과 중간 미션들을 완료하면 된다.

인 식	• Line Scan Camera 를 통해 직선, 곡선 및 스쿨존 인식 • Infra Sensor 를 통해 장애물 인식 • Encoder 를 통해 현재 차량 속도 인식
판 단	• 도로 상황에 맞춰 Line Keeping 을 위한 Servo 조향 값 판단 • School Zone 진입 및 탈출 판단 • School Zone 내에서 장애물 회피 상황 판단 • School Zone 외에서 급제동 상황 판단
주 행	• 도로 상황에 맞춰 Line Keeping 주행 • School Zone 진입 및 탈출 시 차량 속도 제어 • School Zone 내에서 장애물 인식 시 장애물 회피 주행 (차선변경) • School Zone 외에서 장애물 인식 시 긴급 제동(AEB)

2 설계 내용

2.1 하드웨어 구성

2.1.1 하드웨어 구성도

시스템의 전체적인 하드웨어 구성도는 다음과 같다.

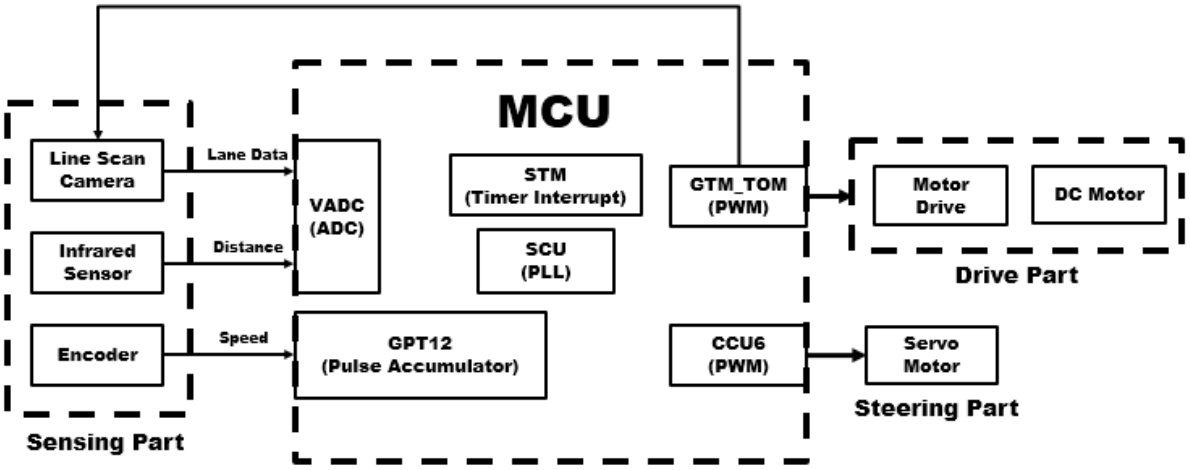


Fig 2.1.1 Hardware Architecture

MCU 는 차선 정보를 위한 Line Scan Camera, 장애물 정보를 위한 Infrared Sensor, 현재 주행 속도를 위한 Encoder 를 입력으로 받고 DC 모터와 Servo 모터에 출력을 내뱉는다. MCU 의 기본 기능들 중 우리는 크게 6 가지 Module 을 사용하였다. 보드의 성능을 향상시키기 위한 PLL(Phase-Locked Loop)을 만들기 위한 SCU(System Control Units), 주기적인 Timer Interrupt 를 만들기 위한 STM(System Timer), Analog 값을 Digital 값으로 받기 위한 VADC(Versatile Analog-to-Digital Converter), PWM 을 만들기 위한 GTM(Generic Timer Module)과 CCU6(Capture/Compare Unit 6), 그리고 Encoder 의 Pulse 개수를 카운트하기 위한 GPT12(General Purpose Timer Unit) 기능들을 사용하였다.

2.1.2 외관 구성

전체적인 외관은 다음과 같다. 포맥스 판을 이용하여 회로기판, MCU 등을 올려놓는 받침으로 제작하였다. 먼저 엔코더를 구동 기어 부분과 맞물려 놓고 그 위에 회로기판과 모터 드라이버 및 MCU(Infineon Tricore Board) 를 올려놓았다. 더불어 가장 위에는 차선 인식을 위해 카메라를 고정시켜 놓았고 장애물 인지를 위해 적외선 센서를 차량의 전면 부에 배치했다. 초기에는 카메라가 차량 중심에서 앞쪽 부분에 위치해 있었지만 선회 시 더 향상된 라인 인식을 위해 차량의 중심에 가깝게 위

치를 변경했다. 또한 카메라가 차량 진동으로 인하여 흔들리게 되면 차선 인식에 악영향을 미칠 수 있어 스페이서와 프레임을 통해 단단히 고정하였다.

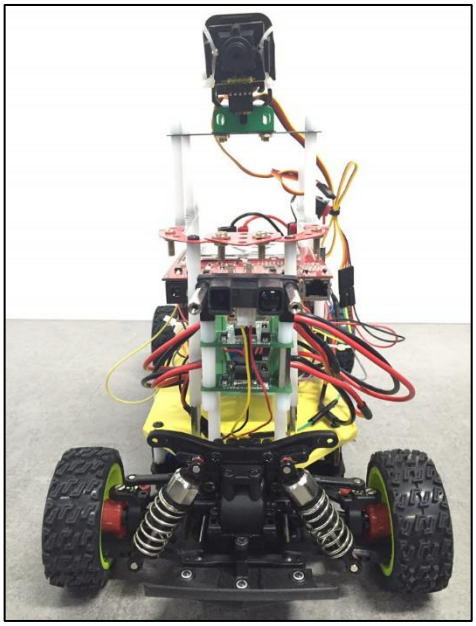


Fig 2.1.2.1 정면

Fig 2.1.2.2 후면

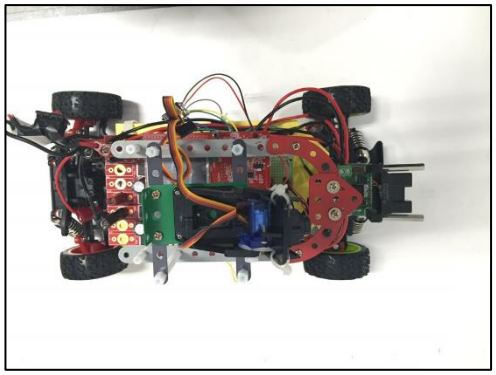


Fig 2.1.2.3 위

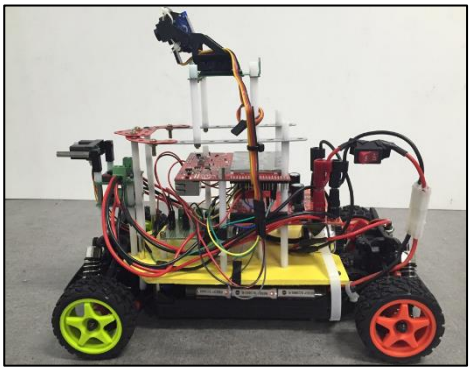


Fig. 2.1.2.4 측면

2.1.3 전원부

전원 : 7.2V 3000mAh Ni-MH

우선 배터리 전원 부에서 7.2V의 전압을 Motor Driver에 직접 공급하고, 이를 통해 DC 모터를 구동하였다. 또한, Servo motor는 6V 에서 최대 토크를 내기 때문에 DC-DC Converter를 통해 7.2V를 6V로 감압한 뒤 공급하였다. 더불어 다른 DC-DC Converter를 통해 7.2V를 5V로 감압한 뒤 MCU와 각종 센서(Encoder, Line Scan Camera 및 Infra Sensor)에 전원을 공급하였다. 아래의 사진은 6V, 5V 전압 변환 모듈에 관한 하드웨어 사진이다.

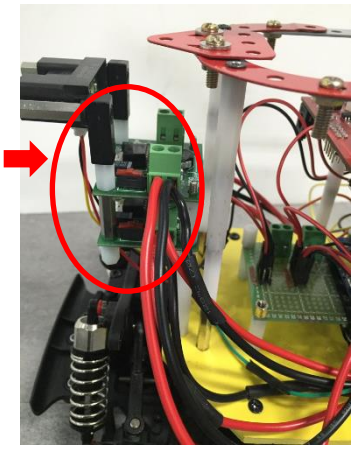


Fig 2.1.3.1 5V 전압 변환

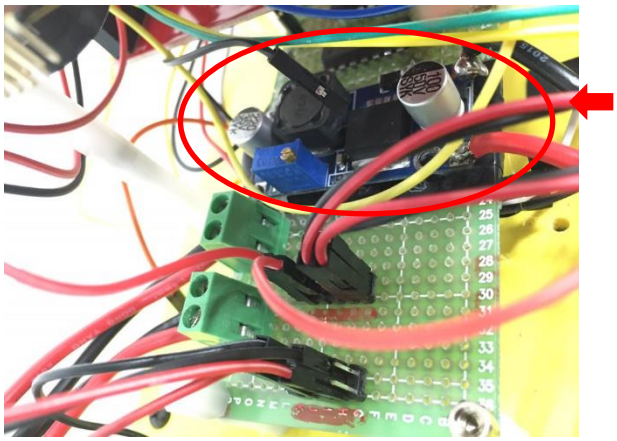


Fig 2.1.3.2 6V 전압 변환

2.1.4 조향부

Servo 모터에서 동력을 전달하는 혼 부분 및 링키지 부분의 하드웨어 적인 간극을 최소화하고자 케이블 타이 및 순간 접착제를 이용하여 단단히 고정하였다. 링키지 부분에서 간극이 존재하였기 때문에 좌측에서 우측으로, 우측에서 좌측으로 서보 Duty 를 변경했을 때 돌아가는 조향 각의 크기가 달랐다. 이 부분은 소프트웨어 적으로 보정하는 과정을 거쳤다.

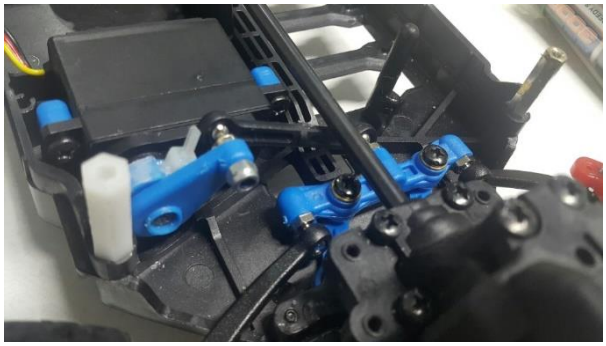


Fig 2.1.4 조향부

2.1.5 센서부

차선과 장애물 인식을 위하여 2 개의 입력 센서를 사용하였다. 라인스캔 카메라는 회전 중에도 안정성을 확보하기 위해 회전 중심에 가까운 차체 중앙에 설치하였고. 적외선 센서는 장애물을 보다 멀리서 인지 하기 위하여 차의 맨 앞부분에 설치하였다.

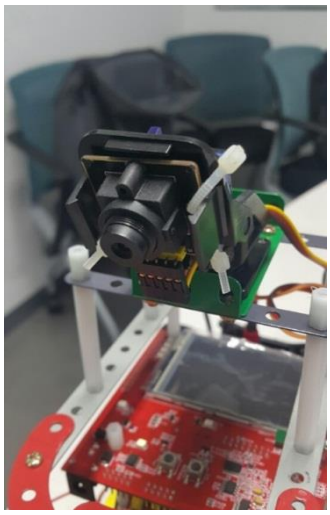


Fig 2.1.5.1 Line scan camera

Fig 2.1.5.2 Infrared Sensor

2.1.6 구동부 및 회로 구성

DC motor 제어를 위해서는 Encoder 를 통한 모터의 회전 수 측정이 필수 적이다. 따라서 차의 후면에 있는 기어에 Encoder 를 맞물리는 형식으로 설치 하였다. 또한 주행 중 Encoder 가 흔들릴 수 있기 때문에 케이블 타이를 통해 고정 시키었다.

Fig 2.1.6.3, Fig 2.1.6.4 는 Encoder 를 통해 모터의 정•역 방향을 구분해서 신호를 받기 위한 목적의 회로이다. D-Flip Flop 과 And-Gate 를 이용하여 Encoder 신호의 정•역을 구분하였고 And-Gate 를 이용하여 이를 각각의 신호로 받을 수 있었다.

Fig 2.1.6.1 DC motor

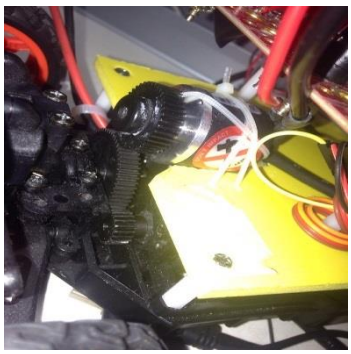


Fig 2.1.6.2 Rotary Encoder

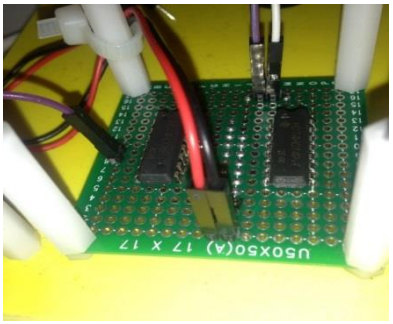


Fig 2.1.6.3 Encoder 정•역 회로

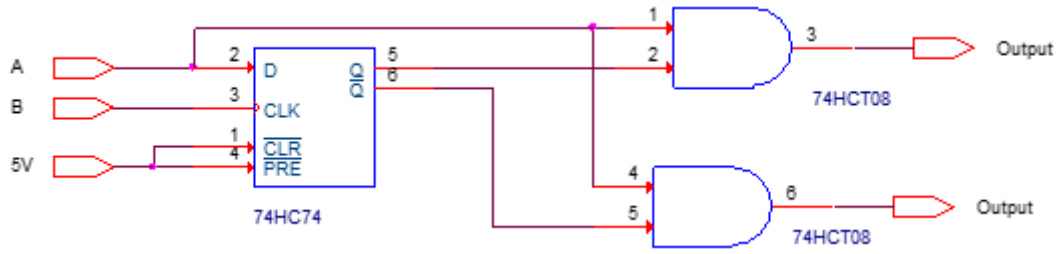


Fig 2.1.6.4 Encoder 정•역 회로

2.2 소프트웨어 구성

2.2.1 Camera Signal Processing

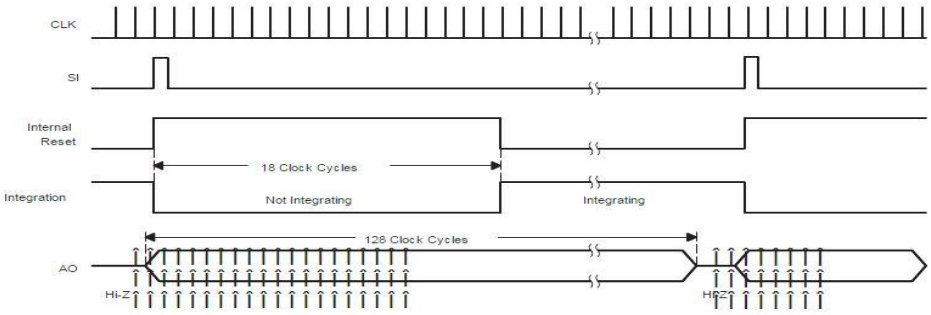


Fig 2.2.1.1 Camera Data Sheet

TSL- 1401 라인 스캔 카메라는 Fig 2.2.1.1 에서 보이는 것처럼 CLK 와 SI 를 인풋으로 넣어주면 카메라 내부적으로 처리과정을 거치고 AO 에서 우리가 원하는 출력이 나온다. 다음 그림은 입력과 출력에 대해 좀더 자세한 내용을 나타낸다.

	PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t w(H)	Clock pulse duration (high)		50			ns
t w(L)	Clock pulse duration (low)		50			ns
t s	Analog output settling time to ± 1%	R L = 330 Ω, C L = 50 pF		350		ns

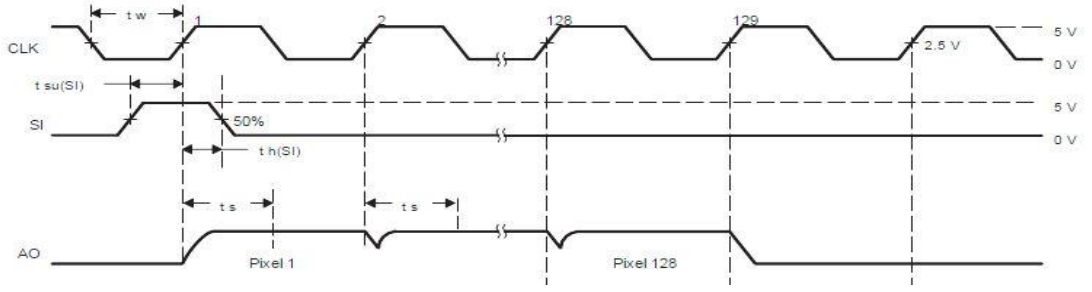


Fig 2.2.1.2 Camera Data Sheet

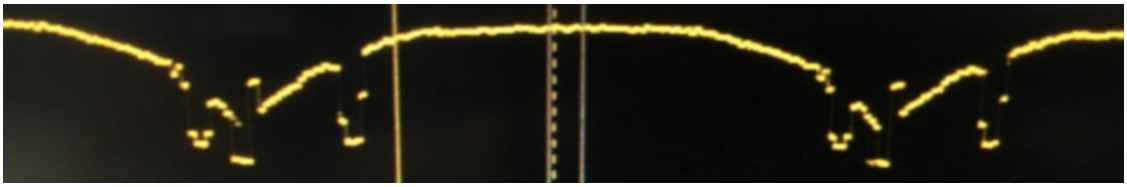


Fig 2.2.1.3 Oscilloscope Data of Camera

Fig 2.2.1.2 에서 보면 CLK 의 Duty 는 50%로 설정하게끔 되어있고 SI 는 첫 CLK 가 High 인 구간 중에 Low 로 설정되게 되어있다. 그리고 128 개의 Pixel 정보가 모두 출력되면 그 뒤의 CLK 신호에 대해서 AO 는 더 이상 반응하지 않는다. 우리 팀 역시 위 매뉴얼 대로 사용하려 했지만 문제가 있었다. 위 매뉴얼 에서는 CLK 와 SI 의 시작 Timing 이 다르지만 이 점을 처리하기에 다소 어려움이 있었기 때문이다. 우리는 CLK 와 SI 의 시작 Timing 이 같았기에 SI 의 Duty ratio 를 더 크게 주는 방법으로 이를 극복했다. 원하는 SI 와 CLK 를 넣어준 다음 AO 를 오실로스코프로 찍어보면 Fig 2.2.1.3 와 같이 나타난다.

2.2.2 Line detect

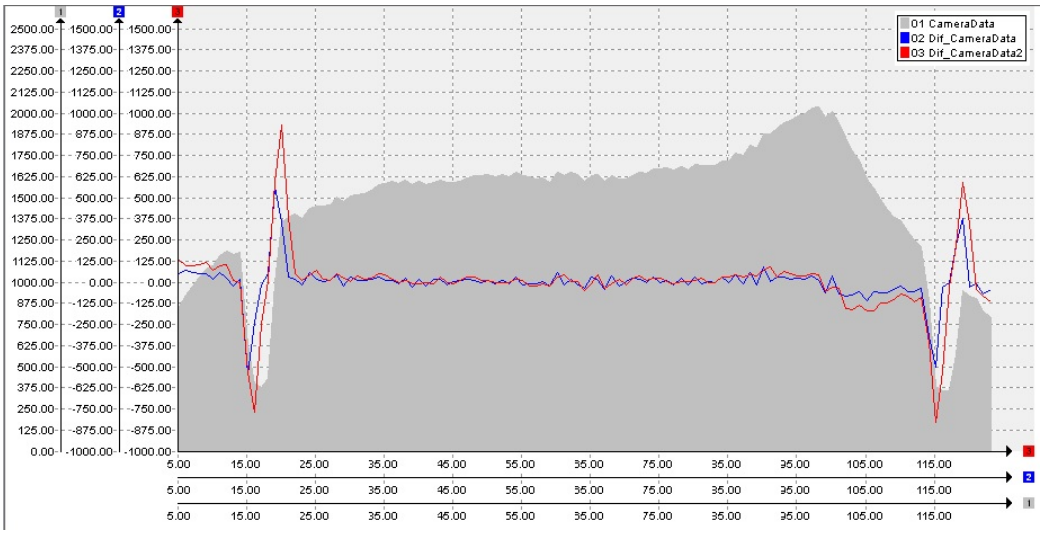


Fig 2.2.2 Camera data

TSL-1401 Line Scan Camera 는 128 개의 라인 Pixel 데이터를 3.3V 의 Analog 데이터로 출력한다. 보드의 VADC 모듈을 이용해 값을 출력하면 회색 배경과 같이 나타난다. Line 의 정보를 얻기 위해 Pixel 간의 차분을 이용하면 파란색 선과 같이 나타난다. Pixel 이 담고 있는 정보를 살펴보면 검은색 부분이 얼마나 들어와 있는지를 갖고 데이터를 출력하게 된다. 일반적으로 Pixel 의 간격과 우리가 보고자 하는 라인의 경계 부분이 정확히 일치할 수 없기에 라인의 경계 부분 에서는 흰색 – 흰색과 검은색 – 검은색 순으로 Pixel 데이터가 출력될 것이다. 즉 차분을 하나의 Pixel 이 아닌 2 개의 Pixel 로 한다면 더 극명하고 정확한 값을 얻을 수 있을 것이고 이를 표시한 선이 빨간 선이다. 카메라의 시야를 조금 더 넓게 쓰고자 라인 경계의 안쪽 부분을 사용하기로 했다. 위 빨간 선에서 중앙을 기준으로 왼쪽에

서 가장 큰 값이 왼쪽 Edge 가 되고 오른쪽 부분에서 가장 작은 값이 오른쪽 Edge 가 된다.

2.2.3 Corner detect

코너와 직선 구간별 유리한 게인 값 적용이 가능하다면 훨씬 부드럽고 빠른 주행이 가능할 것이다. 이를 가능케 하기 위해선 직선 구간과 코너 구간의 구분이 필요하다고 판단했다. 현재 카메라의 Edge 판단은 중앙을 기준으로 좌측이 왼쪽 Edge 우측이 오른쪽 Edge 가 된다. 중앙을 기준으로 설정하였기에 왼쪽 차선이 우측을 침범하거나 우측 차선이 왼쪽으로 침범하게 되면 차선 고유의 정보는 잃어버리게 된다. 즉 왼쪽 차선이 카메라 중앙 부분을 넘어서 우측에 찍히게 된다면 이 차선은 앞으로 우측 차선이 된다. 위와 같이 차선이 카메라 중앙을 넘어서 침범하는 구간은 코너에서 밖에 나타나지 않기에 우리는 이러한 상황이 나타나는 구간을 코너 구간이라고 설정하였다. 그리고 차선 침범과 동시에 코너 구간이 판별되면 침범된 차선을 원래 라인의 차선으로 인식해 주는 보정 알고리즘을 추가하였다.

2.2.4 구간별 서보 게인 값 설정

2.2.4.1 직선 구간 게인 값

직선구간에서는 중앙을 맞추어 달리는 게 바람직하다. 따라서 카메라가 중앙에 놓여 있을 때의 R_M과 현재 카메라가 바라보고 있는 중앙 C_M의 차이에 gain값을 곱하여 서보를 조향한다.

$$ServoAngle = (R\_M - C\_M) * P$$

하지만 위와 같이 조향할 시 다음과 같은 오실레이션이 발생할 수 있다..

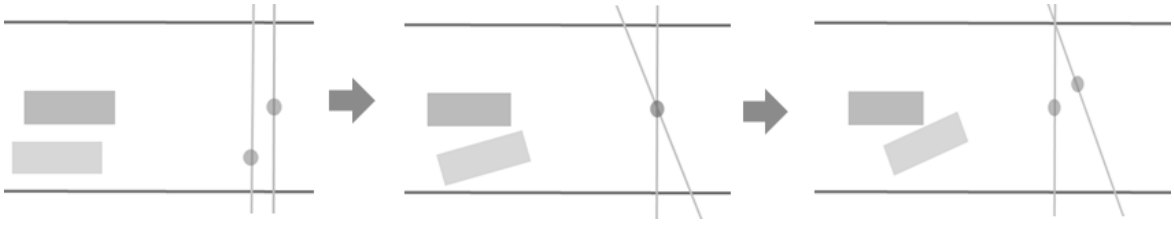


Fig 2.2.4.1.1 Oscillation Case

위와 같은 상황이 있다고 보면 초기에는 R_M과 C_M의 차이만큼 왼쪽으로 조향한다. 그렇게 주행하던 중 카메라의 중앙이 기존의 중앙과 일치하게 되면 직선으로 조향 값을 조정한다. 다시 카메라의 중앙과 기존의 중앙의 차이가 오른쪽에 발생하면서 오른쪽으로 조향을 하게 된다. 위와 같은 상황이 반복적으로 일어나면서 오실레이션이 발생한다.

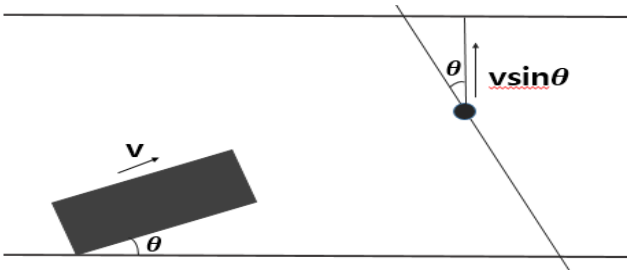


Fig 2.2.4.1.2 Camera middle point change with angle

모형 자동차가 라인과 평행한 방향으로 주행한다면 카메라 중앙값의 변화는 없을 것이다. 하지만 모형 자동차가 일정한 각도만큼 회전한 상태에서 주행한다면 중앙값은 변할 것이다.
θ만큼 틀어진 상태에서 직진 한다면 중앙값의 변화는 vsinθ만큼 나타난다. 이 변화량에 저항하는 값을 추가하면 더욱 좋은 주행이 가능할 것이다. D gain을 취하면 직선에 큰 회전 각도로 주행로 중앙에 접근하는 것을 막을 수 있다.

$$ServoAngle = (R\_M - C\_M) * P + (C\_M)' * D$$

추가적으로 조향 값에 오차의 크기에 따라 P게인을 다르게 준다면 다음과 같다.

$$P_1 = 1\ (|R\_M - C\_M| < 3)$$
$$P_2 = 1.2\ (3 \leqq |R\_M - C\_M| < 5)$$
$$P_3 = 1.4\ (5 \leqq |R\_M - C\_M| < 8)$$
$$P_4 = 1.7\ (8 \leqq |R\_M - C\_M| < 10)$$

위처럼 P gain을 Pixel간의 차이가 클수록 더욱 크게 준다면 더욱 빠르고 부드럽게 중앙에 도달할 수 있다.

2.2.4.2 코너 구간 게인 값

코너 구간에서는 직선 구간과 다르게 변화에 민감해야 한다. 따라서 직선 구간과 다르게 D gain이 존재하는 것은 바람직하지 않다. 또한 회전 중에는 일정한 차이를 유지하는 것이 바람직하다. 그 만큼의 차이를 일정하게 설정할 수 있도록 다음과 같이 P게인을 설정했다.

$$ServoAngle = (R\_M - C\_M) * P$$

$$P_1 = 2\ (|R\_M - C\_M| < 3)$$

$P_2 = 2.5 \ (3 \leqq | R\_M - C\_M | < 5)$

$P_3 = 2 \ (5 \leqq | R\_M - C\_M | < 8)$

$P_4 = 1.5 \ (8 \leqq | R\_M - C\_M | < 10)$

위와 같이 주행 테스트를 했을 경우 보다 원하는 회전 반경으로 부드럽게 코너 구간을 주행할 수 있었다.

2.2.5 AEB, School zone 알고리즘

어린이 보호 구역은 카메라 픽셀 값들의 변화 정도를 누적하여 판단하게 했으며 적외선 센서를 이용해 장애물 회피와 AEB(Autonomous Emergency Braking)을 구현하였다. 스쿨 존 내부에서 장애물과 마주할 경우 장애물 회피 알고리즘을 적용하였고 스쿨 존 외부에서 장애물을 만날 경우에는 AEB 를 적용하였다. 장애물 회피 알고리즘의 경우 현재 자신의 차선 데이터를 저장하고 Follow the carrot method(Fig.2.2.5.1)를 이용해 차선 변경을 하였다. 또한 차선 변경 시 생기는 Heading angle error(Fig 2.2.5.2)를 해결하기 위하여 모형 차량을 Bicycle model(Fig 2.2.5.3)로 가정하고 차량의 속도와 차선 변경에 걸리는 시간, 현재 조향 각도를 바탕으로 보상해주었다. 이를 통하여 차선 변경 시 생기는 오실레이션을 조기에 잡을 수 있었다. AEB 의 경우는 Motor 와 Encoder 를 사용해 PI 제어를 하여 Slip 을 최소화하여 깔끔하게 정지하도록 설계하였다.

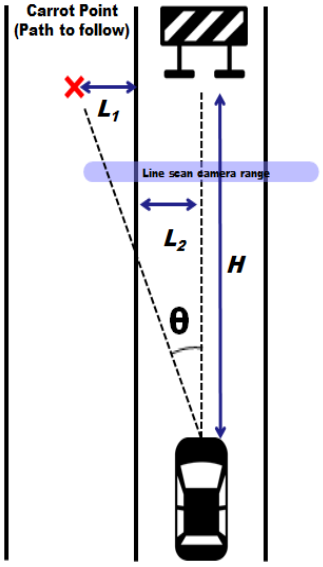


Fig 2.2.5.1 Follow the carrot method

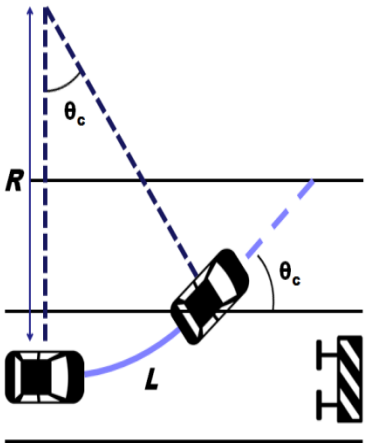


Fig 2.2.5.2 Heading angle error compensation

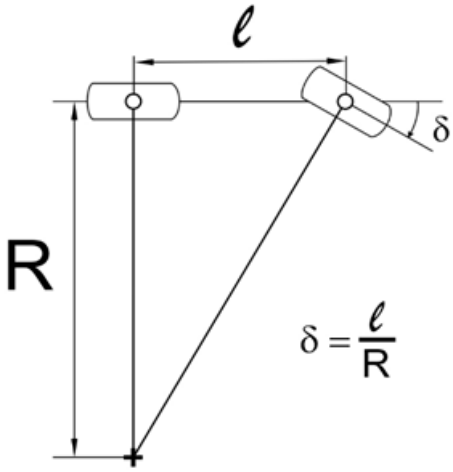


Fig 2.2.5.3 Bicycle model

3 주요 장치 이론 및 적용 방법

3.1 TDD (Test Driven Development)

3.1.1 TDD 개요

TDD 는 테스트 주도 개발로 구체적인 알고리즘을 작성하기 전에 테스트 코드를 먼저 작성하고 이것을 만족하는 실제 코드를 작성하는 것이다

Fig 3.1.1.1 기존의 소프트웨어 제작 방식

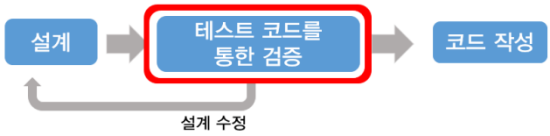


Fig 3.1.1.2 TDD 를 이용한 제작 방식

기존의 프로그래밍 기법은 위의 Figure 3.1.1.1 과 같이 개발을 마친 후 테스트 케이스들을 직접 적용하는 방식으로 개발이 진행되었다. 이 방식은 원하는 결과를 얻지 못하면 설계 단계에서부터 크게 수정을 해야 할 가능성이 있었다. 이는 설계에 대한 검증이 충분하지 않았기 때문에 고려하지 못한 부분들을 발견하기 때문이다. 이러한 개발 상의 문제들을 해결하기 위해서 제안된 방법이 바로 TDD 이다. 이 밖에도 많은 장점을 가진 TDD 는 자바(JAVA)와 파이썬(Phyton)과 같이 객체 지향 언어를 사용하고 소프트웨어 중심의 테스트 환경 구축이 쉬운 곳에 국한되어 적용되고 있는 실정이다.

우리 팀은 이러한 TDD 를 지능형 모형차 대회와 같은 Embedded System 개발에 적용해보고자 하였다. 특히 Embedded System 개발의 경우 개발 환경(Host System)과 작동 환경(Target System)이 다르기 때문에 디버깅에 시간을 많이 소모하게 된다. 따라서 TDD 는 실제 Target System 이 아닌 Host System(PC)에서 검증해볼 수 있기 때문에 디버깅 시간도 크게 단축할 수 있었다. 또한 여럿이서 협업을 하는 상황이었기 때문에 각자 자신 만의 알고리즘을 테스트 해본 후 결과와 함께 이를 검증하고 토의를 진행할 수 있어서 막연한 시행착오를 막을 수 있었다.

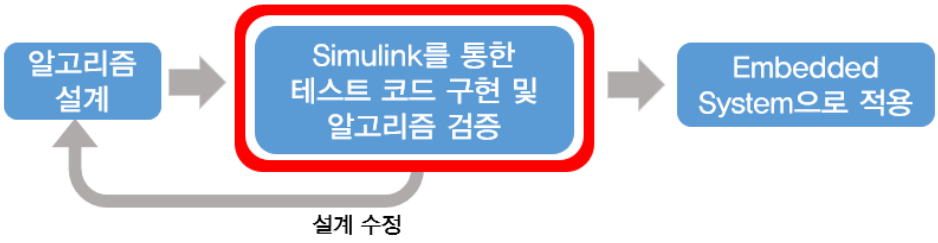


Fig 3.1.1.3 MATLAB 및 Simulink 를 활용하여 TDD 를 구현 및 적용한 Embedded System 설계 과정

3.1.2 지능형 모형차를 위한 TDD 구현

우리 팀은 TDD 의 구현을 Simulink 를 활용하여 비교적 쉽게 해낼 수 있었다. Simulink 상에서 블록 다이어그램을 통하여 전체 지능형 모형차를 위한 알고리즘을 작성하였고, 실제 Input 으로 받게 되는 Port(카메라 센서 데이터, 적외선 센서 데이터) 부분을 테스트 블록(MATLAB 으로 구현한 가상 트랙)으로 대치해서 우리 팀의 알고리즘에 통과시킨 후, 시뮬레이션 기능을 통하여 그래프를 확인해 봄으로써 검증할 수 있었다.

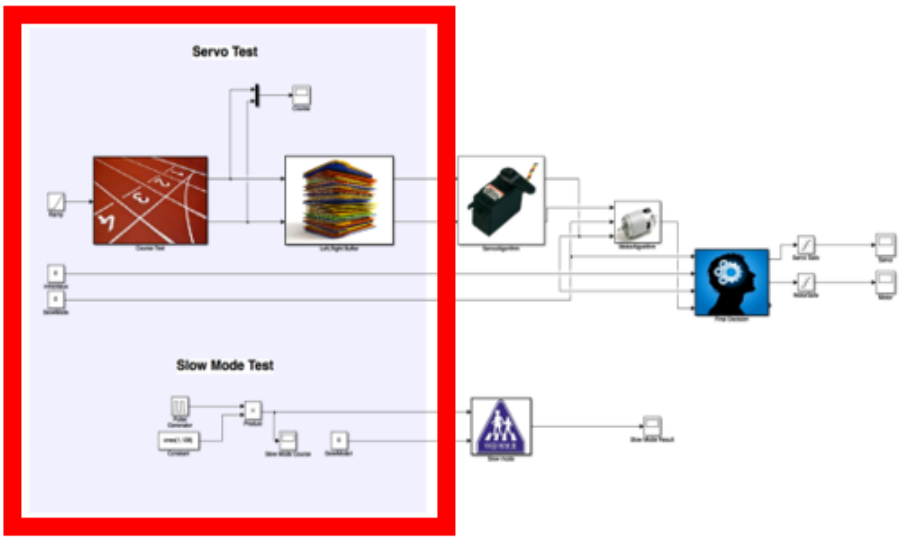


Fig 3.1.2.1 Simulink 를 이용한 지능형 모형차 블록 선도 및 TDD 대치영역(붉은 영역)

시뮬레이션을 이용해 나오는 결과 그래프의 분석을 통해 알고리즘의 성공, 실패 여부, 그리고 제어의 경향성을 볼 수 있었다. 실제 환경에 적용해보면 차량의 제원과 외부 환경에 의해 시뮬레이션과는 조금 다른 결과가 나타나기도 했지만, 기본적인 알고리즘 검증에 있어 실제로 많은 시간을 단축할 수 있었다

- TDD 적용사례 : 카메라 값 테스트 코드를 통한 코너 서보 조향 알고리즘 검증

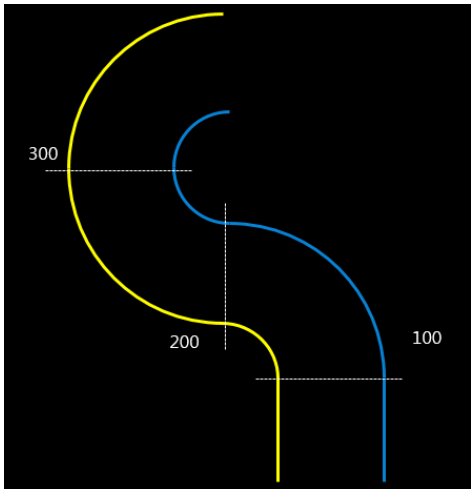


Fig 3.1.2.2 실제 트랙 모습

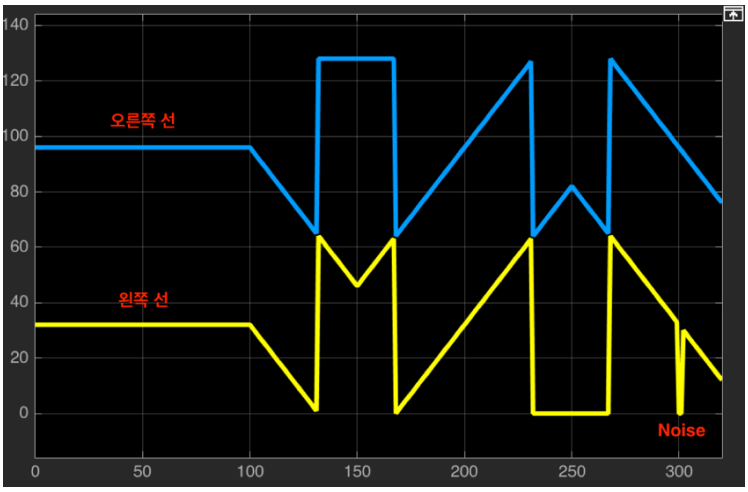


Fig 3.1.2.3 카메라가 인식하는 Left, Right 트랙 데이터

먼저 Fig 3.1.2.2, 3.1.2.3 과 같이 차량의 주행 상황에 따른 라인 인식 테스트를 작성하였다. Fig 3.1.2.2 는 실제 주행 코스이고 Fig 3.1.2.3 은 주행 중의 Line Scan Camera 가 인식하는 라인 데이터를 보여주고 있다. 카메라 데이터는 설계 단계에서 값이 어떻게 들어오는지 눈으로 확인하고 예측하여 작성하였다. 이를 통하여 직진 구간을 통과 후 곡선 구간에 진입했을 때의 라인 인식 데이터를 실제 주행 전에 고민하고 코너 상황에서 발생하는 카메라 데이터의 변화 등을 사전에 이해하여, 더 효율적인 알고리즘을 생각해 내는데 도움을 받을 수 있었다.

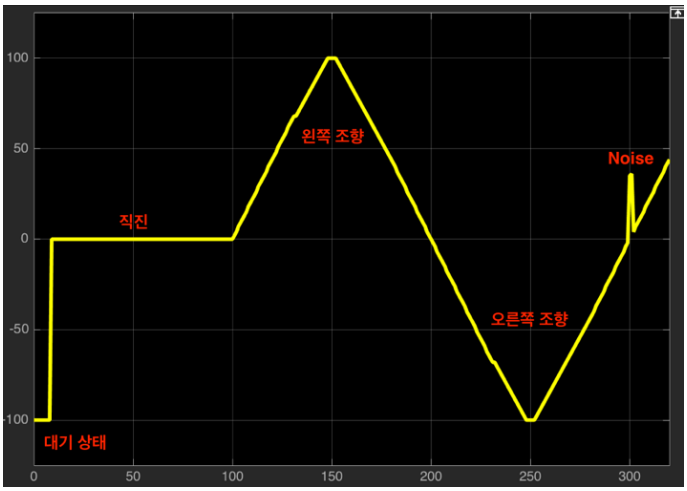


Fig 3.1.2.4 Servo Result of Test

시뮬레이션 트랙 테스트를 통해 위와 같은 서보 조향 값 결과를 얻을 수 있었다. 대기 상태는 초기 세팅을 위한 준비 구간이고, 출발 이후 직선, 곡선 코스를 통과하는 서보 조향 값을 보임을 알 수 있다. 마지막으로 Noise 에 대한 대응을 보기 위하여 위와 같이 튀는 값을 일부러 적용하였는데 (Fig 3.1.2.3), 테스트 과정에서 이를 해결하기 위해 처음에는 Low Pass Filter 를 적용하였으나 이는 라인 변화에 따른 반응성을 오히려 둔하게 하는 영향을 끼칠 수 있음을 알게 되었다. 대신에, 노이즈의 원인인 라인 검출 알고리즘을 더욱 강인하게 보강하였다. 이처럼 Simulink 내에서 시스템을 구현하고 시뮬레이션을 진행함으로써, 단시간 내에 실제 모형 차량을 통하여 직접 검증하지 않고도 알고리즘을 크게 개선할 수 있었다.

3.2 PID 제어

PID 제어기를 설계하는 과정에서 각 P, I, D 게인 값을 찾는데 있어서 Trial and error 를 통하여 게인 값을 찾으려면 오랜 시간이 걸린다. 하지만 이 과정에서 제어 대상(모터)의 모델링이 되어있다면, Simulink 에서 제공하는 PID Tuner 를 이용하여 단시간에 적절한 게인 값을 찾을 수 있다. 단순화된 DC Motor 의 모델링을 위해서는 J, b, K, R, L 와 같은 모터 Parameter 들의 값을 알아야 한다. PID Tuner 를 활용하여 게인 튜닝의 과정을 줄이기 위하여 우리 팀은 MATLAB 에서 제공하는 Parameter estimation 기능을 이용하여 DC motor 의 모델링을 진행하고, PID tuner 를 이용하여 PID 게인 값을 결정하였다. 우선 Figure 19 과 같이 Simulink 를 이용하여 DC Motor 를 구성하였다. 또한

Eclipse 를 통하여 얻은 속도 데이터를 MATLAB 상의 Workspace 에 불러온 뒤, Parameter estimation 기능을 이용하여 속도 데이터에 가장 근접한 Curve 를 Fitting 하였다(Fig 3.2.2 좌). 이로부터 추정된 모터 Parameter 들(Fig 3.2.2 우)을 얻을 수 있었다.

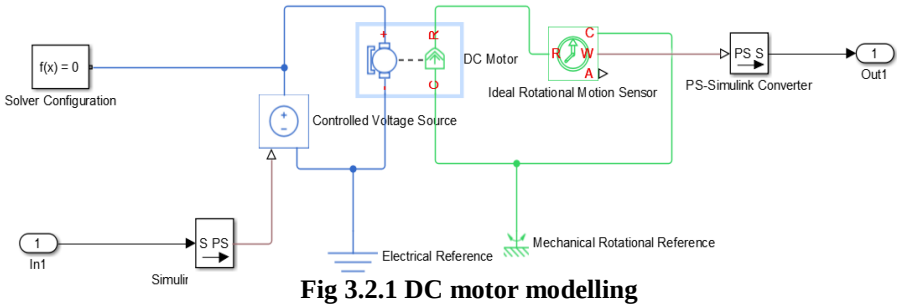
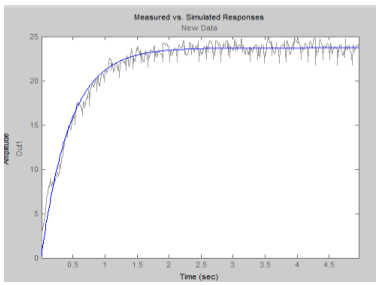


Fig 3.2.1 DC motor modelling



Name	Value	Min	Max
J	0.0107	0.0107	0.0107
K	0.0538	0.0538	0.0538
L	8.3682e-04	8.368...	8.368...
R	0.1525	0.1525	0.1525
b	0.0048	0.0048	0.0048
tout	<100x1 double>	0	10

Fig 3.2.2 DC motor parameter estimation

그 이후, PID 제어기를 설계 한 후, Simulink 의 PID Auto tuner 을 이용하여 최적의 P, I, D 게인 값을 손쉽게 얻을 수 있었다. (Fig 3.2.3)

실제적으로 Eclipse 에서 받은 모터 속도 데이터는 공중에 띄운 상태로 측정한 데이터여서 실제 주행에 최적화된 PID 게인 값과는 약간의 차이가 있었다. 하지만 MATLAB 에서 추정한 게인 값과 큰 차이가 없었기에 많은 시간을 단축할 수 있었다.

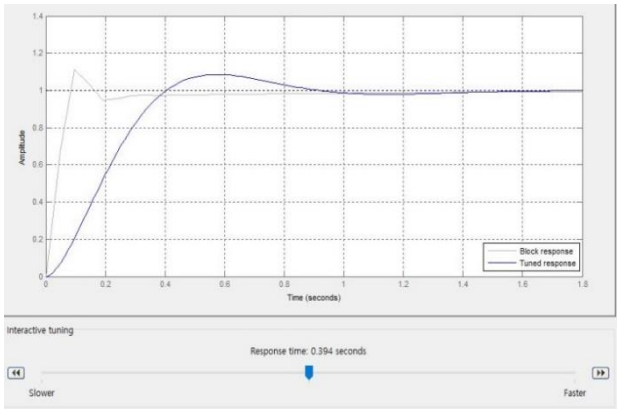


Fig 3.2.3 PID Auto tune

4. 결론 및 토의

4.1 결론

대회를 준비해 나아가면서 완성도가 높아짐에 따라 주행 알고리즘의 문제보다는 라인 인식 문제가 더욱 크게 다가왔다. 우리가 Test 하는 환경과 대회장의 환경이 일치하지 않기에 다양한 환경에서 Test 를 해보았고 그에 따른 빛의 밝기를 고려하기 위해 어떠한 Threshold 를 설정하여야 하나가 가장 중요한 문제였다. 같은 주행 알고리즘의 같은 Track 이라도 다른 환경에서 완주를 못하는 현상을 자주 보였다. 차선 인식을 어떻게 할 것이냐에 대한 문제는 우리 팀의 핵심 문제였고 이 문제를 Threshold 를 여러 값들을 참조해 유동적으로 변하게 하는 방법으로 해결했다.

또한 지금까지 있었던 대회와는 다르게 이번 대회는 2 개의 Line 에서 우측 차선을 유지하면서 주행해야 하는 대회였다. 2 개의 Line 이 존재했기에 우측과 좌측 커브의 회전반경이 달랐다. 특히 우측 커브의 회전반경이 너무 작았기에 다양한 문제점이 발생했다. 커브에 미리 대응하기 위해 카메라를 멀리 보게 설정하면 커브 주행 시 현재 Line 이 아닌 다른 곳을 바라보았고, 카메라를 가까이 보게끔 해 커브 주행 Line 을 보게끔 하면 코너 진입에 대한 대응이 느려져 속도를 올리지 못했다. 우리 팀은 더 빠른 속도를 원했기에 카메라를 더욱 멀리 보게 하는 방법을 선택했고 주행 시 카메라가 다른 곳을 보는 문제를 Algorithm 으로 해결했다.

모터를 제어하려면 Encoder 로부터 값을 읽어 PI 혹은 PID 제어가 필수적이다. 우리 팀도 모터를 제어하기 위해 Encoder 값으로부터 PID 계수를 산출해내 적용하였다. 하지만 문제점이 발생하였다. Tricore TC-237 보드에서 GTM 을 사용하여 모터 PWM 을 만들 경우 PWM 의 Duty 를 변화시키는 Trigger 를 자주 사용할 경우 PWM 의 출력이 우리가 기대하는 일정한 PWM 이 나타나지 않다는 것이다. Trigger 를 사용하여 Duty 20%를 반복적으로 요구한 결과와 Trigger 없이 단 초기의 Setting 으로 Duty 20%를 설정한 결과는 압도적으로 Trigger 가 없는 PWM 세팅이 좋았다. 우리 팀은 필요한 구간에만 제어를 사용하기로 결정하였다. 가속 상황은 주로 직선 구간에서 이루어졌기에 정확한 속도가 크게 중요하지 않았지만 감속 상황은 코너 진입 상황이나 AEB 등 보다 정해진 속도에 빠르게 도달해야 하는 구간이었다. 따라서 우리는 가속 시에는 모터 제어를 하지 않고 정해진 PWM 을 한번 입력하였고 감속 시에는 제어를 하기 위해 Encoder 값을 참조하여 짧은 주기로 제어를 사용했다. 그 결과 직선 주행시의 부드러운 장점과 감속상황에서의 필요한 만큼의 빠른 감속 모두 취할 수 있었다.

4.2 토의

대회를 준비하면서 초기에 2D 카메라를 사용하려는 시도를 했었다. Infineon 에서 올려준 ILLD 파일도 공부해보고 Manual 도 공부해 보았지만 통신을 성공하지 못해 결국 Line scan 카메라로 변경했다. 2D 카메라를 통해 상황을 인식했다면 보다 많은 정보로 최단거리 혹은 차선 인식 알고리즘이 더욱 강건

할 수 있었을 거라 생각되는데 이점이 매우 아쉽다. 또한 대회를 준비하면서 PID Gain 혹은 서보 모터의 Gain 등 다양한 값들을 실험으로 찾아야 했다. 이 문제도 역시 통신이 되지 않아 매번 Usb cable 을 보드와 연결해 새로운 값을 연결해 주어 Gain 값을 변경해주어야 했고 그 결과 많은 시간이 소요되었다. 통신의 문제를 초기에 해결하였다면 더욱 원활한 대회 준비가 되었을 거라 생각된다.

초기에 MCU 를 지급받고 부품을 사고 기본 Setting 을 하기까지 처음 다뤄보는 기능들과 부품들이 많았다. MCU Manual 은 학생들이 읽기에 다소 어려운 부분이 많았고 이를 공부하는데 많은 시간이 소요되었다. 이러한 문제로 차가 완성되기까지 긴 시간이 소요되어 주행 Test 를 하는 시간이 충분하진 못했지만 처음 접하는 MCU 에 대해 우리 스스로 공부해보고 도전해 보았던 시간들을 팀원들은 매우 의미 있게 생각하고 있다.

Appendix – 부품 목록

제조사	부품명	수량	사용목적
Towerpro	MG996R	1	Servo 모터
LK Embedded	LK-DMS-PRO	1	장애물 감지용 적외선 센서
Parallax	TSL 1401	1	라인 스캔 카메라
Autonics	E30S4-1000-3-V-5	1	Encoder
Any Vendor	74HCT08N	1	Encoder 방향 판단 AND 게이트
Any Vendor	74HC74	1	Encoder 방향 판단 D-Flip Flop
Reedy	WolfPack 3000 Battery Pack (Ni-MH 7.2V)	1	전원 Battery
Any Vendor	LM2596	1	Step Down Regulator
Roboblock	LM2575	2	5V Regulator