2016 지능형모형차 경진대회 MathWorks 특별상 부문

2016
지능형 모형차 경진대회
MATLAB 활용 보고서

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

1 개요

1.1 참가 배경

2016 년 세계 경제 포럼에서 자율주행자동차가 10 대 미래 기술로 선정 되었을 정도로 자율주행자동차에 대한 관심은 뜨겁고, 실 도로에서 주행할 날이 머지 않았다. 자율주행자동차의 시대가 도래함에 따라 Embedded system 을 설계하고 무인 자동차를 제어할 수 있는 인재에 대한 수요는 꾸준히 증가하고 있다. 이에 우리들은 학교에서 배운 제어 이론들과 역학, 회로이론 지식들을 바탕으로 센서들을 통해 외부 정보를 수집하고 MCU 를 통해 모형자동차를 제어하여, 향후 실제 차량의 적용 방향에 대해 이해하고 나아가 여러 문제점들을 해결해 보고자 대회에 참가하게 되었다.

1.2 설계 목표

지능형 모형자동차 대회의 예선은 스피드 경주이고, 본선은 AEB(Autonomous Emergency Braking)와 School zone, 장애물 회피 및 차선 변경의 미션 또한 수행하여야 한다. 이를 위해서는 탄탄한 하드웨어 설계와 사용자가 시스템을 쉽게 파악할 수 있고, 오류를 최소화 할 수 있는 소프트웨어 설계는 필수적이다. 이를 위한 일환으로 MATLAB 을 적극 활용하였다. 특히 모형차의 전체 소프트웨어 시스템을 설계하고 목표를 달성하기 위한 알고리즘을 적용하는 과정에서 MATLAB Simulink 의 Code generation, Monitor 등 여러 기능들과 내장함수들을 사용하였다.

2 MATLAB 을 활용한 소프트웨어 설계 및 하드웨어 성능 개선

2.1 제어 이론 및 MATLAB 을 이용한 제어 알고리즘 구현

차량의 레지스터를 직접적으로 관계하는 기능을 제외한 전체 차량 알고리즘을 MATLAB/Simulink 를 이용해 작성했으며 Code Generation 기능을 적극 사용해 Eclipse for Tricore 와 MATLAB, 두 IDE 의 결과물을 쉽게 연결하고 모듈화 및 협업의 효율을 극대화 하는 방식을 사용하였다. 또한 소프트웨어 디자인 패턴으로 TDD(Test Driven Development) 방식을 적용하였다. TDD 는 현재 소프트웨어를 개발하는데 있어서 전세계적으로 가장 즐겨 사용하는 개발 방법론 중 하나 이다. 가상의 상황에서 미리 시뮬레이션 해볼 수 있는 기능을 통해 MCU 에 코드를 적용하여 디버깅하는 단계를 최소화 하였고 개발 시간을 크게 단축할 수 있었다.

2.2 TDD(Test Driven Development)

2.2.1 TDD 개요

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

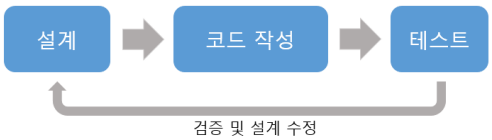


Figure 1. 기존의 소프트웨어 제작 방식

Figure 2. TDD 를 이용한 제작 방식

기존의 프로그래밍 기법은 위의 Figure 1 과 같이 개발을 마친 후 임의의 테스트 케이스들을 입력하여 확인하는 방식으로 개발이 진행되었다. 이 방식은 원하는 결과를 얻지 못하면 설계 단계로 돌아가 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.

TDD 는 위의 Figure 2 와 같이 코드 작성에 앞서 테스트 코드 작성을 하게 된다. 이 테스트 코드는 알고리즘을 검증하는 것으로 에러나 버그가 없는 코드를 작성해야만 성공적으로 실행할 수 있다. 이러한 테스트 코드를 작성하는 과정에서 설계 시 고려되어야 하는 기능들과 요건들에 대해 다시 한번 재점검할 수 있다. 또한, 설계 단계에서부터 결과에 대한 고민이 수반되기 때문에 개발 도중에 큰 수정 사항 없이 원활하게 개발을 진행할 수 있다.

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.

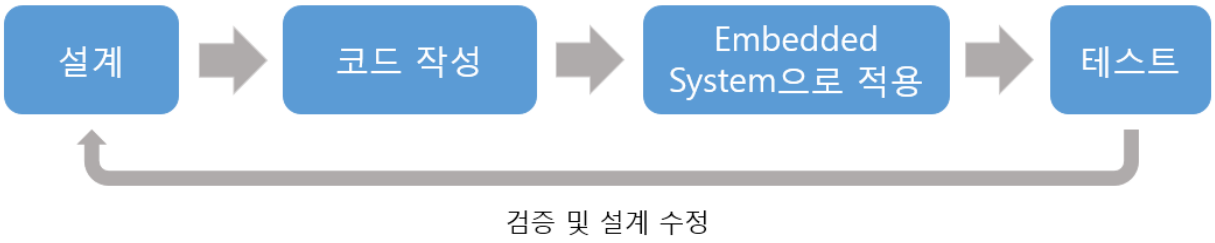


Figure 3. 일반적인 Embedded System 설계 과정

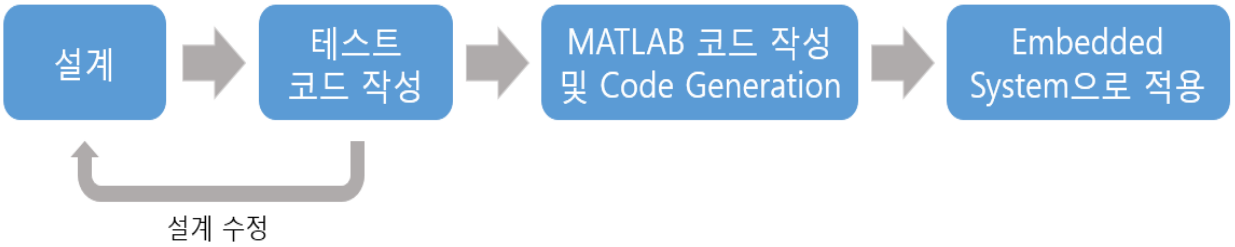


Figure 4. MATLAB 및 Simulink를 활용하여 TDD를 구현 및 적용한 Embedded System 설계 과정

2.2.2 MATLAB을 이용한 지능형 모형차를 위한 TDD 구현

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

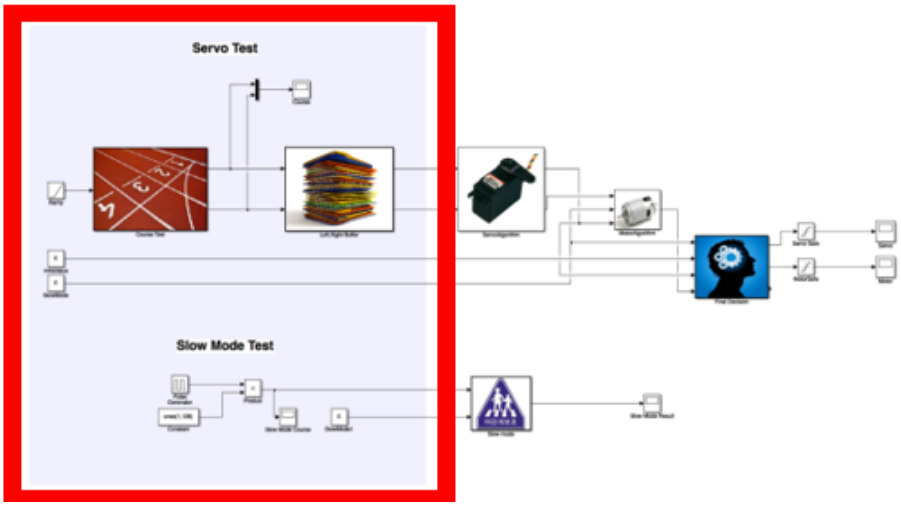


Figure 5. Simulink를 이용한 지능형 모형차 블록 선도 및 TDD 대치영역(붉은 영역)

Simulink 의 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.

또한, Simulink 내의 블록 다이어그램을 통하여 전체 시스템을 구성하였기에, 팀원들이 전체적인 시스템을 파악하기가 쉬웠고, 한 기능에 대하여 Input, Output 이 명확한 하나의 블록으로 구현되어 코드의 수정이 용이하였다.

- **TDD 적용사례 1. 카메라 값 테스트 코드를 통한 코너 서보 조향 알고리즘 검증**

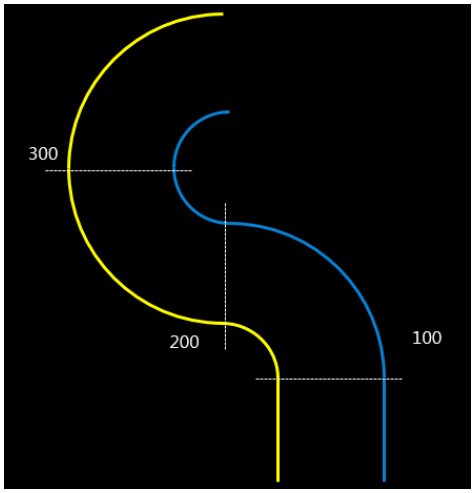


Figure 6. 실제 트랙 모습

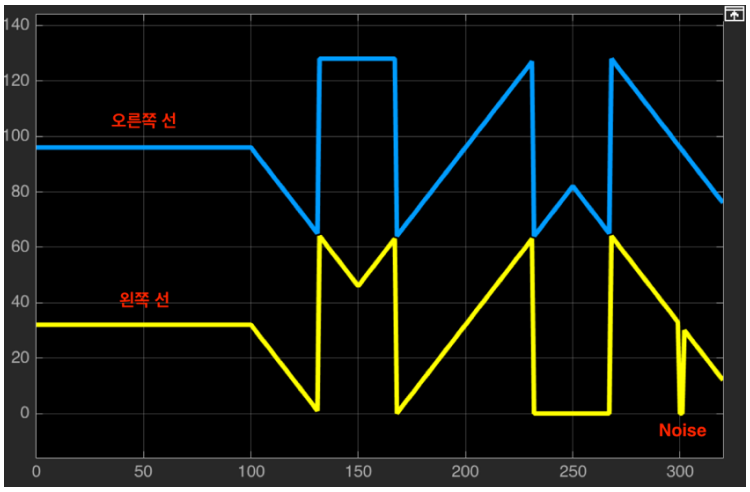


Figure 7. 카메라가 인식하는 Left, Right 트랙 데이터

먼저 Figure 6, 7 과 같이 차량의 주행 상황에 따른 라인 인식 테스트를 작성하였다. Figure 6 은 실제 주행 코스이고 Figure 7 는 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.

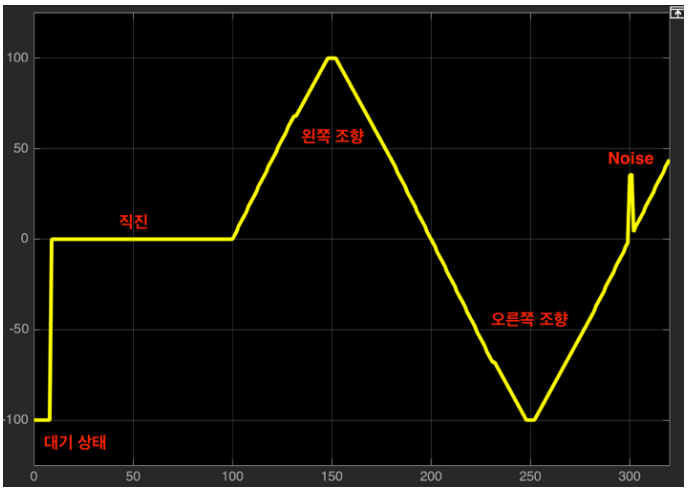


Figure 8. Servo Result of Test

Figure 7 과 같은 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(Figure 7), 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.

- **TDD 적용사례 2. School Zone 인식 테스트**

이 후 School Zone 인식에 관한 테스트를 생성하였다. 이것은 성공 실패 여부를 판단하기 위한 테스트 케이스가 필요했다. School Zone 의 조건인 검은 선을 카메라 값의 급격한 변화를 이용해서 인식할 것이었으므로 흰색을 2000, 검은색을 0 으로 설정하고 급격하게 변화하는 구간을 여러 번 설정하여 School Zone 인식의 신뢰성을 높이고자 했다.

Figure 9. School zone 실제 트랙

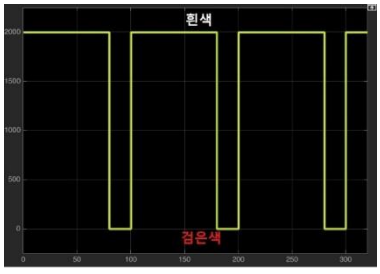


Figure 10. Simulink 를 이용한 테스트 트랙

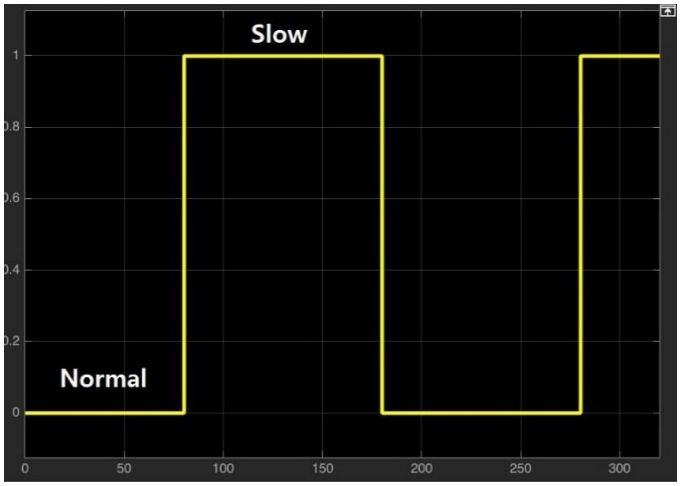


Figure 11. School Zone Result of Test

테스트 모듈의 Output 과 알고리즘의 Input 을 연결하여 MATLAB 상에서 시뮬레이션을 하면 Figure 11 과 같이 원하는 School Zone Result 값을 얻을 수 있다.

성공, 실패 여부가 분명하게 드러나는 테스트 이기에 쉽게 값의 의미를 파악할 수 있었다. 실제와 달랐던 점은, 카메라 데이터 내에 노이즈가 많이 존재한다는 점과, 라인 스캔 카메라가 정확하게 일직선의 픽셀을 읽는 것이 아니기에 빛 데이터의 변화가 한번에 일어나는 것이 아니라 부분부분 진행된다는 점이었다.

위와 같이 알고리즘을 코드로 작성하기 전에 TDD 를 적용하여 디버깅 시간을 최소화 할 수 있었고 알고리즘 개선에 큰 성과를 얻을 수 있었다.

2.3 전체 시스템 프로세스 및 Code generation

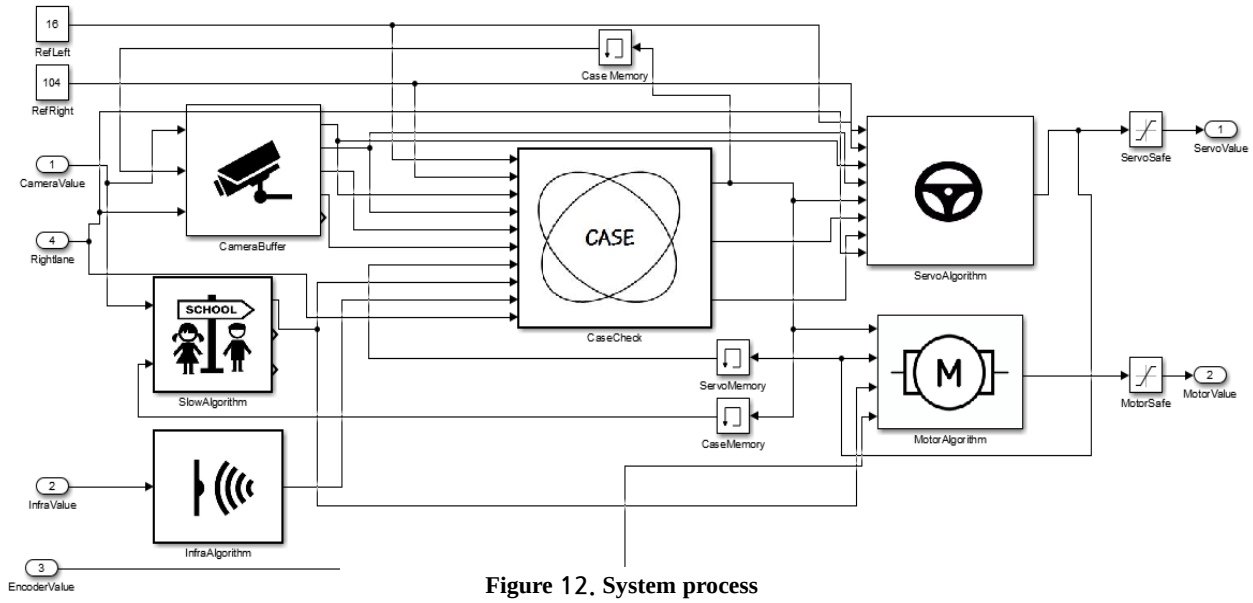


Figure 12. System process

MCU 의 초기 설정과 관련된 하드웨어 관련 코드를 제외한 모든 주행과 관련된 알고리즘은 MATLAB 의 Simulink 를 이용해 작성하였다(Figure 12). Input 은 Camera Value 와 Infra sensor Value 이고, Output 은 Servo Value 와 Motor Value 이다. 그리고 Input 정보를 처리하고, Output 을 만들기 위하여 전체 시스템을 크게 6 개의 모듈로 구성하였다. Camera 정보를 통해 Line 의 정보를 파악하기 위한 Camera Buffer Module, School Zone 을 판단하기 위한 School Zone Module, 장애물을 인지하기 위한 Infra Algorithm Module, 조향 각을 결정하기 위한 Servo Algorithm Module, 모터 속도를 결정하기 위한 Motor Algorithm Module, 그리고 모든 정보를 가지고 현재 주행하는 구간이 직선인지 코너인지 구분하고, School zone 진입 여부를 판단하는 등 모든 주행 상황을 판단하는 Case Check Module 로 구성했다. 이렇게 각 Module 의 목표와 내용을 명확히 구분하고 알고리즘 구성을 진행하였다.

2.3.1 Line searching 알고리즘

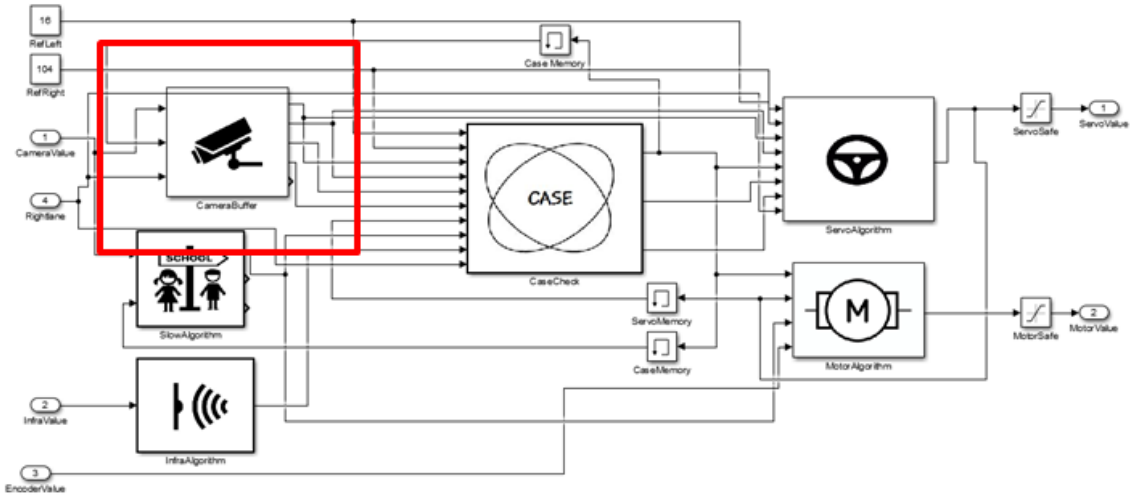


Figure 13. System process : Line searching

인식 알고리즘의 결과는 향후 제어 알고리즘의 결과에 매우 큰 영향을 미치기 때문에 어느 알고리즘보다도 강인한 성능을 보여야 한다. 이를 개선하기 위한 방법의 일환으로 MATLAB 의 내장 함수 Median 을 적용하였다(Figure 15). Median filter 를 통과한 데이터를 인접 픽셀과의 차분 값을 얻어 카메라 픽셀 값의 급격한 변화를 잡아내어 라인으로 인식할 수 있었다. 인식한 라인 값은 Buffer 를 만들어 왼쪽과 오른쪽 선을 10 개씩 저장하였고 주행 시 상황 판단을 하는데 이를 적극 사용하였다(Figure 14). 이와 같이 MATLAB 내장 함수(median)를 사용하여 median 필터를 간단하게 구현했고 이 밖에도 MATLAB 함수들을 사용하여 더욱 간단하게 원하는 것을 구현할 수 있었다.

Line Searching Algorithm

카메라 값을 받아서 **median** 필터로 정제한 후 라인을 검출하고 10개의 **Step** 데이터를 저장해둔다.
흰색선을 발견했을 때를 처리하기 위해 차분의 최대값,최소값의 차이를 출력한다.

****input** : 카메라 값
****output** : 라인 데이터(10개), 차분의 최대최소의 차이

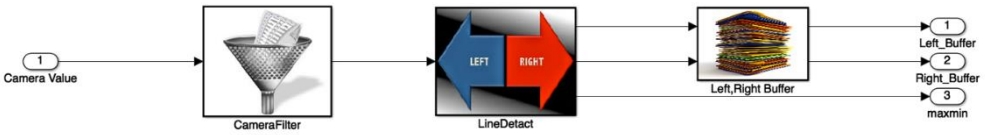


Figure 14. Line Searching Algorithm

```
%% 중간값들을 필터 적용 (5개씩)
for i=3:126
    camera(i)=median([camera(i-2) camera(i-1) camera(i) camera(i+1) camera(i+2)]);
end
```

Figure 15. Median Filter

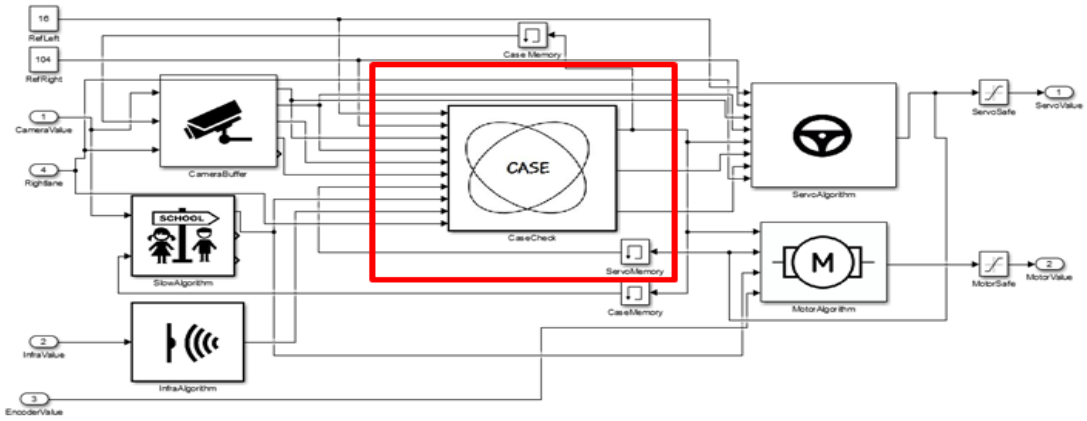


Figure 16. System Process : CaseCheck block

2.3.2 주행 상황(Case)판단 알고리즘

Line searching 알고리즘으로부터 받은 라인 데이터와 적외선 센서로부터 받은 장애물과의 거리를 바탕으로 CaseCheck 블록에서는 주행 상황을 판단한다(Figure 16). Case 는 총 5 가지로 직진 구간, 좌회전 구간, 우회전 구간, 스쿨 존 구간, AEB 구간이다. 직진 구간을 기본 모드로 정하였고, 코너 구간 진입 시 급격히 변하는 라인 정보로 회전 구간을 판별하였다. 예시로 오른쪽으로 커브를 도는 구간에서 카메라 값이 오른쪽 선을 놓치고 왼쪽 선을 오른쪽 선이라고 인식하는 순간을 잡아 우회전 구간이라 인식하고 그 반대의 경우를 좌회전 구간이라 인식하였다. 인식 후에는 다시 직진 구간을 만날 때까지 조향 각도를 크게 유지했다. 스쿨 존 구간을 인식 후 장애물을 만날 시에는 회피 알고리즘을 적용시키었고, 일반 주행 모드에서 장애물을 만날 시에는 AEB 알고리즘을 적용하였다.

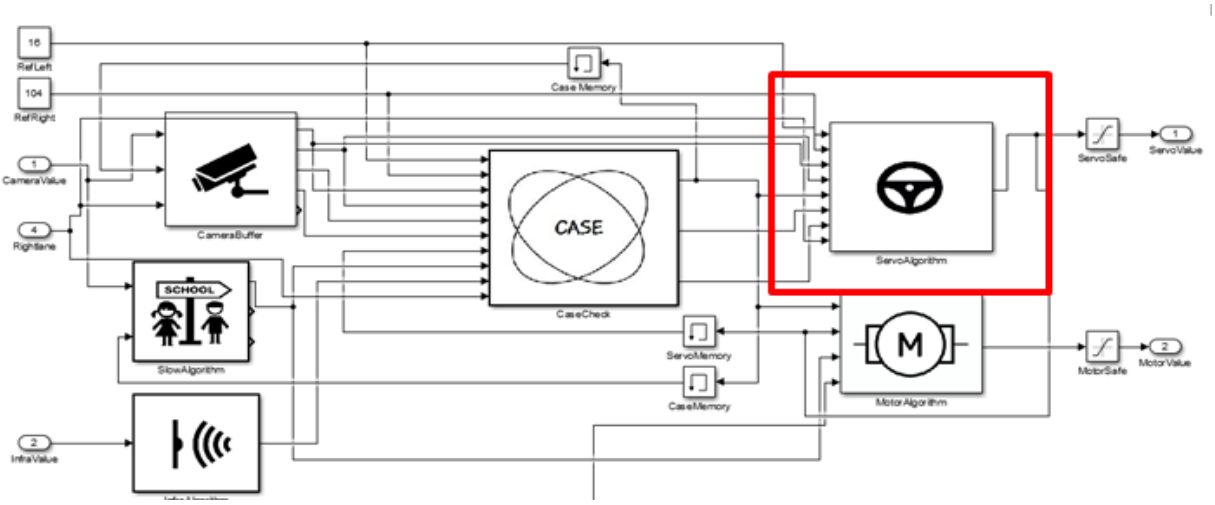


Figure 17. System Process: Servo Algorithm Block

2.3.3 Steering 알고리즘

Figure 18 에서 보이는 Servo Algorithm Block 의 내부는 각 상황에 따른 Servo Value 를 계산하는 알고리즘이 들어있다. Case 1, 즉 직진구간에서는 차량이 중앙에 위치할 때의 라인 정보를 Reference value 로 설정하여 현재 카메라가 보고 있는 라인 정보와 Reference value 간의 오차를 P 제어하여 모형 자동차의 회전 각도를 정하였다. 또한 트랙에서의 오실레이션을 해결하기 위하여 카메라의 범위를 Pixel 단위로 나누어 각기 다른 P value 를 적용하였다. Case 2, Case 3 의 경우는 직진구간보다 회전 각도가 더 크기 때문에 높은 P value 를 적용하였다.

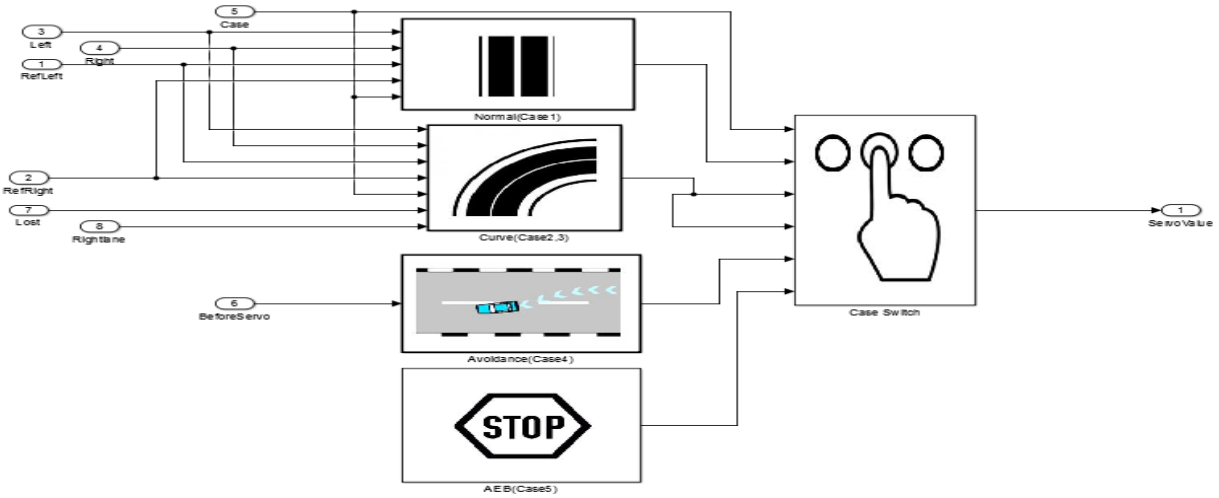


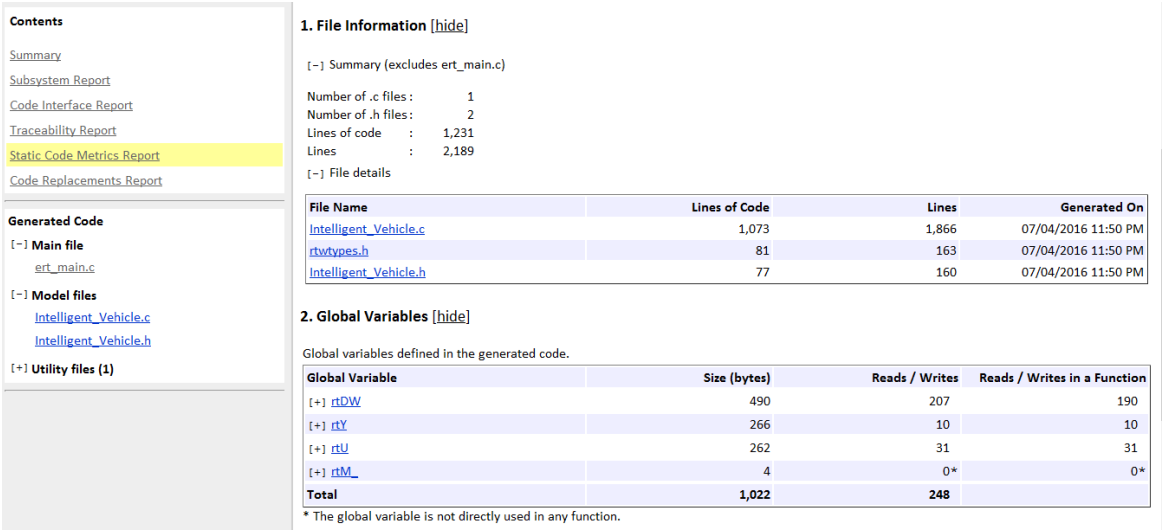
Figure 18. Servo Algorithm Block 내부

2.3.4 Code generation

Figure 19. 전체 프로세스 Block

위의 전체 프로세스를 바탕으로 Simulink 의 Embedded coder code generation 기능을 통하여 Infineon 사의 Embedded board 에 맞는 설정을 선택한 뒤 C 코드를 생성하여 Eclipse 에 이 Source 파일과 header 파일을 추가한 뒤 보드에 전송하였다. 이 후에 위 figure 의 좌측(Input) 데이터를 기존의 Eclipse 상의 레지스터와 연결하였고, 우측(Output) 데이터를 DC 모터와 서보 모터의 PWM 레지스터에 연결해주었다. MATLAB 에서 생성하는 Code Generation 의 리포트는 아래 Figure 과 같다.

Figure 20 에서 볼 수 있듯이 Code Generation 에서 제공하는 전역 변수의 정보와 Source file, Header file 을 통하여 전체 프로세스를 쉽게 파악할 수 있었다. 생성된 Source 코드는 최적화된 코드였기에 사람이 직접 읽어 분석하기엔 오랜 시간이 걸렸다. 하지만 이러한 코드에는 변수를 최소화하여 재활용하고, 각 변수의 overflow 를 방지하는 코드들이 들어있었으며, 수학적인 계산 또한 최적화되어 있었다. 또한 Gain 과 같은 변수는 Initialize 함수에 따로 포함시켜서 Gain 값을 수정하기 용이하게 생성되어 있었다.



Contents

Summary
Subsystem Report
Code Interface Report
Traceability Report
Static Code Metrics Report
Code Replacements Report

Generated Code

[−] Main file
ert_main.c

[−] Model files
Intelligent_Vehicle.c
Intelligent_Vehicle.h

[+] Utility files (1)

1. File Information [hide]

[−] Summary (excludes ert_main.c)

Number of .c files : 1
Number of .h files : 2
Lines of code : 1,231
Lines : 2,189

[−] File details

File Name	Lines of Code	Lines	Generated On
Intelligent_Vehicle.c	1,073	1,866	07/04/2016 11:50 PM
rtwtypes.h	81	163	07/04/2016 11:50 PM
Intelligent_Vehicle.h	77	160	07/04/2016 11:50 PM

2. Global Variables [hide]

Global variables defined in the generated code.

Global Variable	Size (bytes)	Reads / Writes	Reads / Writes in a Function
[+] rtDW	490	207	190
[+] rtY	266	10	10
[+] rtU	262	31	31
[+] rtM_	4	0*	0*
Total	**1,022**	**248**	

* The global variable is not directly used in any function.

Figure 10. Code generation report

2.4 MATLAB 을 통한 하드웨어 성능 개선

모형차에서 주행을 담당하는 DC 모터는 제어이론 적용을 통하여 성능이 개선될 수 있다. 제어 이론으로는 시스템에 대한 명확한 정보 없이도 좋은 제어 성능을 나타내는 PID 제어가 보편적으로 이용 된다. 또한 조향을 담당하는 서보 모터의 경우 서보 Duty ratio 에 따라 조향 각이 결정되는데, 이 두 관계는 일반적으로 선형적이지 않고, 차량의 구동 바퀴와 서보 모터를 연결하는 Linkage 의 영향을 받으므로 이를 고려하여 Duty ratio 에 맞는 조향 각을 찾는 과정이 필요하다. 우리 팀은 DC 모터, 서보 모터의 성능을 MATLAB 및 Simulink 에서 제공하는 기능들을 활용하여 개선하였다.

2.4.1 MATLAB Parameter Estimation 을 이용한 DC 모터 모델링 및 PID Tuning

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 21 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(Figure 22 좌). 이로부터 추정된 모터 Parameter 들(Figure 22 우)을 얻을 수 있었다.

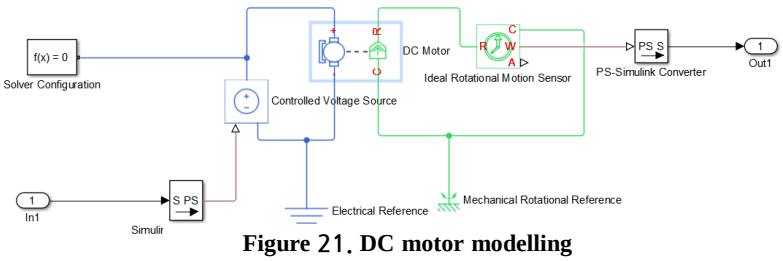
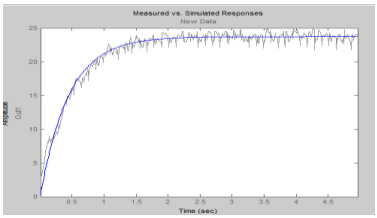


Figure 21. DC motor modelling



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

Figure 22. 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. (**Figure 23)**

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

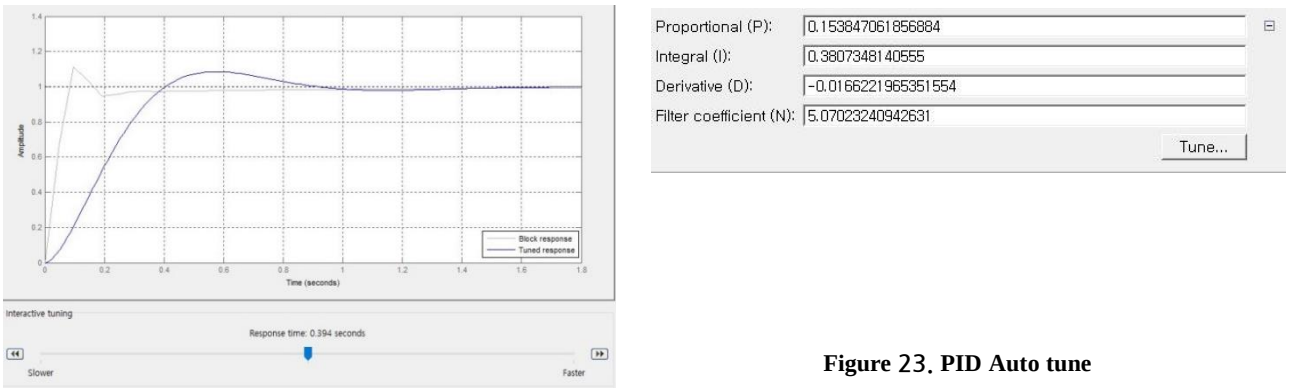


Figure 23. PID Auto tune

2.4.2 MATLAB Curve Fitting 을 이용한 Servo 조향 값 선형 보간(Linear Interpolation) 및 보상 값 결정

PWM Duty 비에 따라서 서보 모터가 조향하는 각도가 결정되는데 이 두 가지는 일반적으로 비선형관계를 띄게 된다. 또한 실제로 본 차량에서는 볼 조인트가 적용되어 있기 때문에 유격이 존재하여, 같은 크기만큼 Duty 비를 적용했을 때 좌측 → 우측으로 회전하여 도달하는 각도와 우측 → 좌측으로 도달하는 각도의 크기가 달랐다. 이러한 문제를 해결하기 위하여, 좌 → 우, 우 → 좌 두 가지 케이스와 서보 Duty 비에 따른 조향 각을 측정하였고, 이 데이터를 MATLAB 을 이용해 분석함으로써 Duty 비에 따른 정확한 조향 값을 도출해내고자 하였다. 좌 → 우, 우 → 좌 데이터를 MATLAB 내의 Curve Fitting Tool 을 이용해 선형 보간한 결과는 다음과 같다.

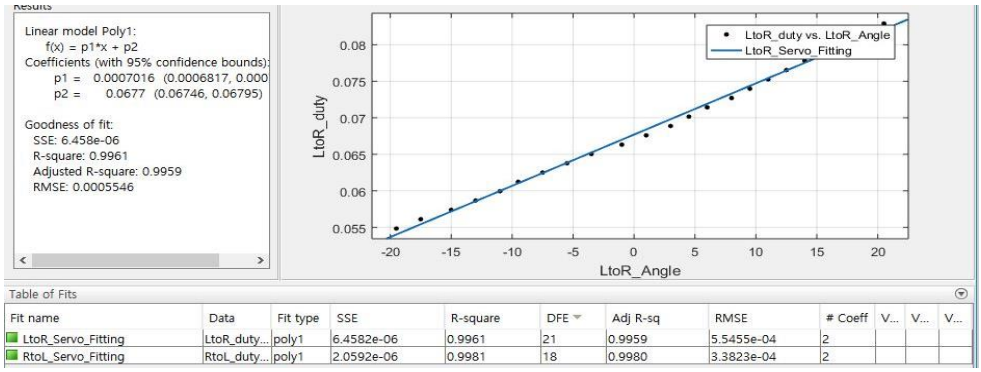


Figure 24. 좌 → 우 방향 Duty 비에 따른 각도 값 선형 보간 결과

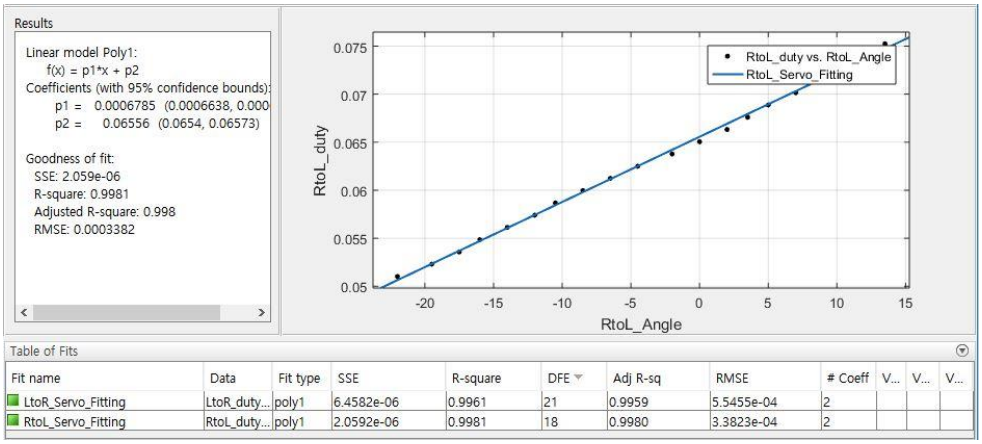


Figure 25. 우 → 좌 방향 Duty 비에 따른 각도 값 선형 보간 결과

Curve Fitting 결과, 두가지 상황에 대하여 1 에 무척 가까운 R-square (결정계수) 및 Adj R-sq(수정 결정계수)를 보임으로써, 두 방향 모두 다 서보 Duty 비에 따른 서보 각도가 선형적인 관계를 나타낸다고 볼 수 있었다.

여기서의 문제점은 Figure 26 와 같이 방향에 따라서 서보 Duty 비 - 조향 각의 크기에 차이가 존재하는 부분 이었다.

이 부분에서 두 직선의 서보 Duty 비 차이 값이 평균적으로 0.0022(0.22%) 가량 차이가 났다. 따라서 우리 팀은 이러한 부분을 고려해주어 이전 조향 값을 저장해 다음에 넣어주어야 할 조향 값이 이전 조향 값보다 우측이면 이 0.22%를 고려한 Duty 비 값을 추가로 더해주어 양쪽의 크기를 맞추어 주었다. 이를 통하여 결과적으로 이전보다 더 나은 서보 성능을 얻어낼 수 있었다.

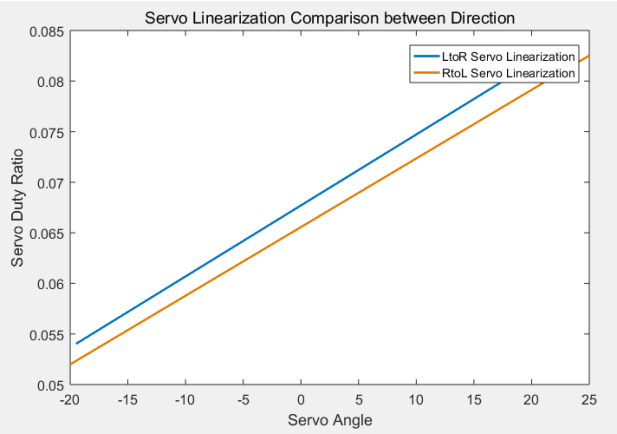


Figure 26. 좌 → 우, 우 → 좌에 따른 서보 선형 보간 결과 비교

3 하드웨어 구성

3.1 하드웨어 구성도

우리 시스템의 전체적인 하드웨어 구성도는 다음과 같다.

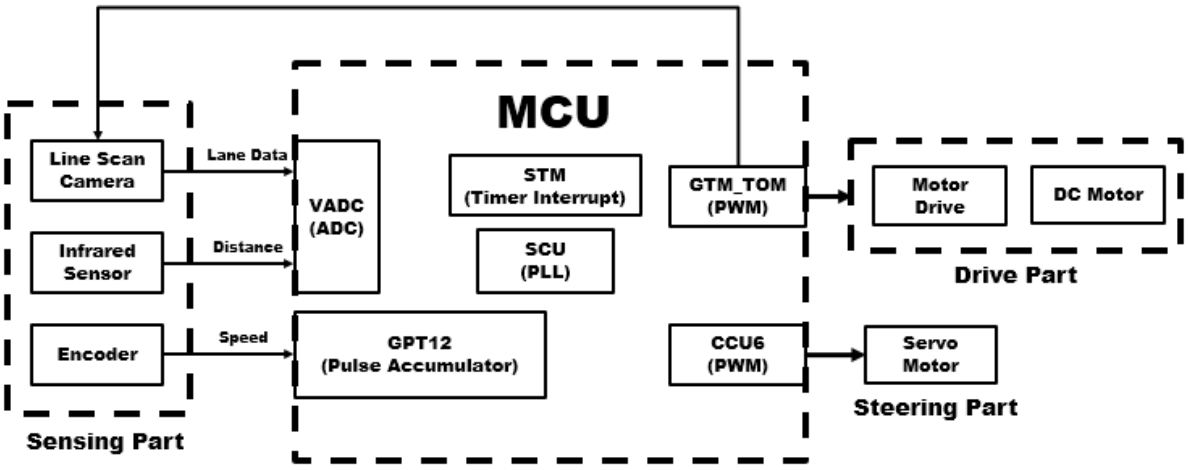


Figure 27. Hardware Architecture

MCU 는 차선 정보를 위한 Line Scan Camera, 장애물 정보를 위한 Infrared Sensor, 현재 주행 속도를 위한 Encoder 를 입력으로 받고 DC 모터와 서보 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)를 사용하였다.

3.2 외관 구성

전체적인 외관은 Figure 31 와 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 위치를 변경했다. 또한 카메라가 차량 진동으로 인하여 흔들리게 되면 차선 인식에 악영향을 미칠 수 있어 스페이서와 프레임을 통해 단단히 고정하였다.

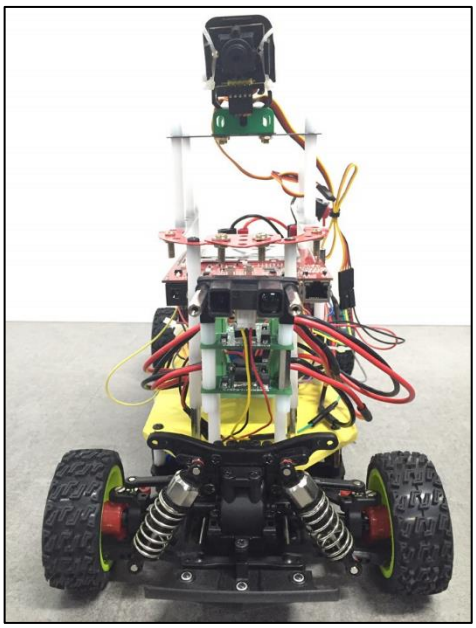


Figure 28. 정면

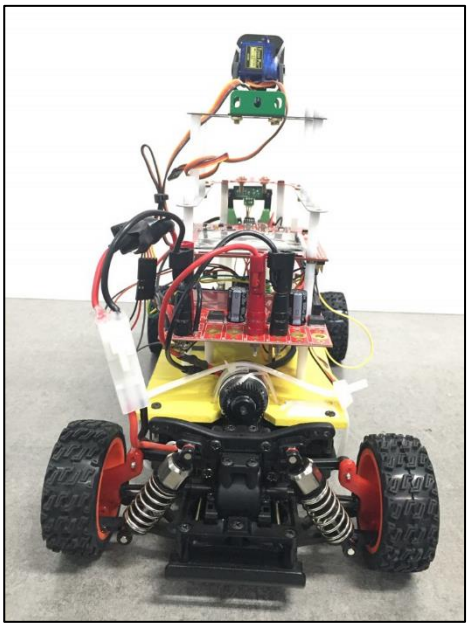


Figure 29. 후면

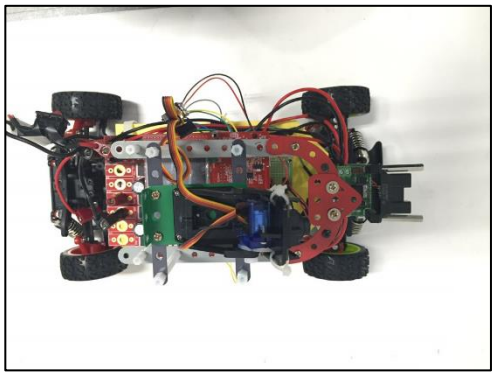


Figure 30. 윗면

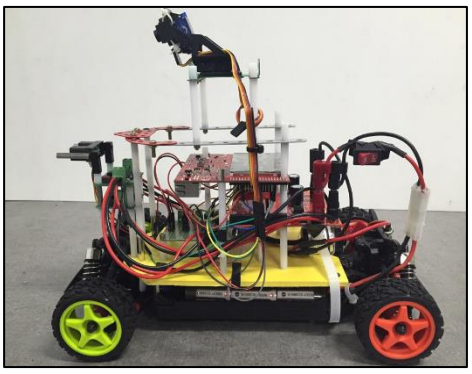


Figure 31. 측면

4 결론 및 토의

4.1 결론

알고리즘 제작의 전 과정을 MATLAB Simulink 를 이용해서 제작해 보았다. Tricore 와 MATLAB 두 IDE 의 연결을 쉽게 하기 위해 전체 모델에서의 Input, Output 변수를 고정하여 간단히 파일 3 개 (Intelligent_Vehicle.c, Intelligent_Vehicle.h, rtwType.h) 를 덮어 쓰는 것으로 바로 실행이 가능하도록 했다. 이와 같은 구성으로 하드웨어 조작 영역과 실제 알고리즘 부분의 영향력을 최소화 할 수 있었다. Block Diagram 형식으로 작성했기에 알고리즘 진행 순서를 직관적으로 알 수 있었고 또한 각각의 Block 들을 모듈화 하여 작성할 수 있었다.

4.2 MATLAB 을 이용한 제작과정을 통해 느낀 점

모형 자동차 제작의 전 과정을 MATLAB Simulink 를 이용해서 만들어 보면서 직접 모든 함수를 작성하고 또 파일을 생성했을 때와는 매우 큰 차이가 있다는 것을 알았다. 사람이 프로그래밍을 하면 실수가 생기기 마련인데 이를 시뮬레이션 상에서 미리 확인하고 또 Code Generation 을 통해 타겟 시스템에 최적화된 프로그램을 생성할 수 있는 것이 매우 매력적이었다. 또한 MATLAB 의 수많은 내장 함수를 사용할 수 있다는 것도 정말 큰 장점이라고 느꼈다. 많은 것을 사용해 보지는 못했으나 만약 거대한 프로젝트를 진행할 때 수많은 내장 함수들을 이용해 빠르고 정확한 프로그래밍을 할 수 있을 거라는 기대가 생겼다.

기존의 사용했던 IDE 들과 다르게 직관적인 연결 관계를 알 수 있는 Simulink 를 통해서 협업을 할 때 그 장점이 극대화 됨을 느꼈다. 개인이 Block 을 새로 작성하고 교체하는 식으로 수많은 테스트를 빠르게 할 수 있었고 알고리즘의 수정 또한 각 Block 사이의 연결 관계만 알고 있다면 쉽게 할 수 있었기 때문이다.

또한 Block 화를 통한 모듈화가 아니라 단순히 C code 를 작성하였다면 논리적 흐름을 항상 염두 하였을 것이고 이에 따라 실수가 더욱 빈번하게 생길 수 있다. 하지만 MATLAB 의 Simulink 상에서 모듈화를 통해 코딩을 진행하게 되어 이러한 고민으로부터 벗어날 수 있었다. 각 Module 의 구간별 Input 과 이를 통한 Output 의 관계만 명확히 이해하면서 코딩을 진행하면 Module 의 독립성을 확보할 수 있었고, 팀원들의 아이디어 혹은 알고리즘이 어느 곳에 위치하여야 하는지 직관적으로 알 수 있었다.

Appendix - 부품 목록

제조사	부품 명	수량	사용 목적
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