**2차 프로젝트 OSEK OS**

IT 집중교육 1



5조

|  |  |
| --- | --- |
| 팀원 | 이름 |
| 1 | 안인수 |
| 2 | 유시현 |
| 3 | 이상현 |

목차

[**1. 개요** 1](#_Toc53176182)

[**2. 차량 1** 2](#_Toc53176183)

[**2-1. 주 기능** 2](#_Toc53176184)

[**2-1-1. 전 / 후진** 2](#_Toc53176185)

[**2-1-2. 좌/우 조향** 2](#_Toc53176186)

[**2-1-3. 속도, 브레이크 모드 변환** 3](#_Toc53176187)

[**2-1-4. 브레이크** 3](#_Toc53176188)

[**2-2. 보조 기능** 3](#_Toc53176189)

[**2-2-1. 좌/우 조향 최대 각 측정 및 정면 축 인식** 3](#_Toc53176190)

[**2-2-2. 후륜 모터 수율 제어** 3](#_Toc53176191)

[**2-3. 추가 기능** 4](#_Toc53176192)

[**2-3-1. 자동 주차** 4](#_Toc53176193)

[**2-3-3-1. 주차 공간 탐색(TaskSearchParkingSpace)** 4](#_Toc53176194)

[**2-3-3-2. 음악 재생** 11](#_Toc53176195)

[**3. 차량 2** 12](#_Toc53176196)

[**3-1. 주기능** 12](#_Toc53176197)

[**3-1-1 전/후진** 17](#_Toc53176198)

[**3-1-2. 좌 / 우 조향** 17](#_Toc53176199)

[**5. 맺음말** 22](#_Toc53176200)

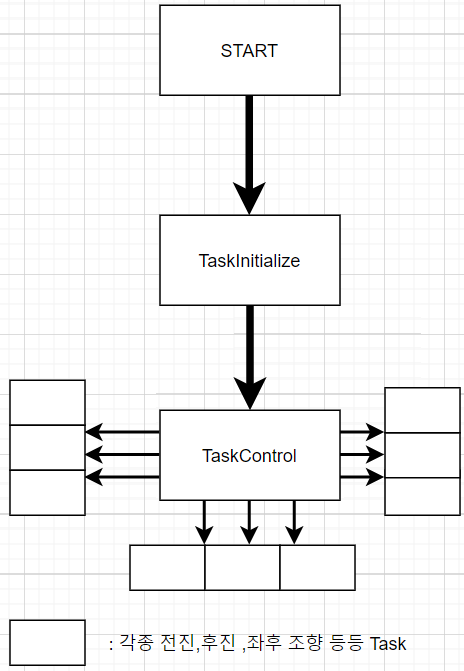
# **1. 개요**

본 프로젝트에서 우리는 Lego Mindstorm과 OSEK OS를 이용하여 실제 차량의 가/감속 장치, 스티어링 장치, 주행보조장치에 대해서 블루투스를 이용한 Guide Car(차량 1), 초음파 센서를 이용한 Follower Car(차량 2)를 구현하였다.

차량 1에서 목표 삼아 중점적으로 설계했던 부분은 부드러운 감/가속을 통해 실제 차량처럼 움직일 수 있도록 하는 것과 추가 기능이다. 폐루프 설계 방식을 통해 어떤 상황에서도 같은 성능을 기대할 수 있도록 구현했다.

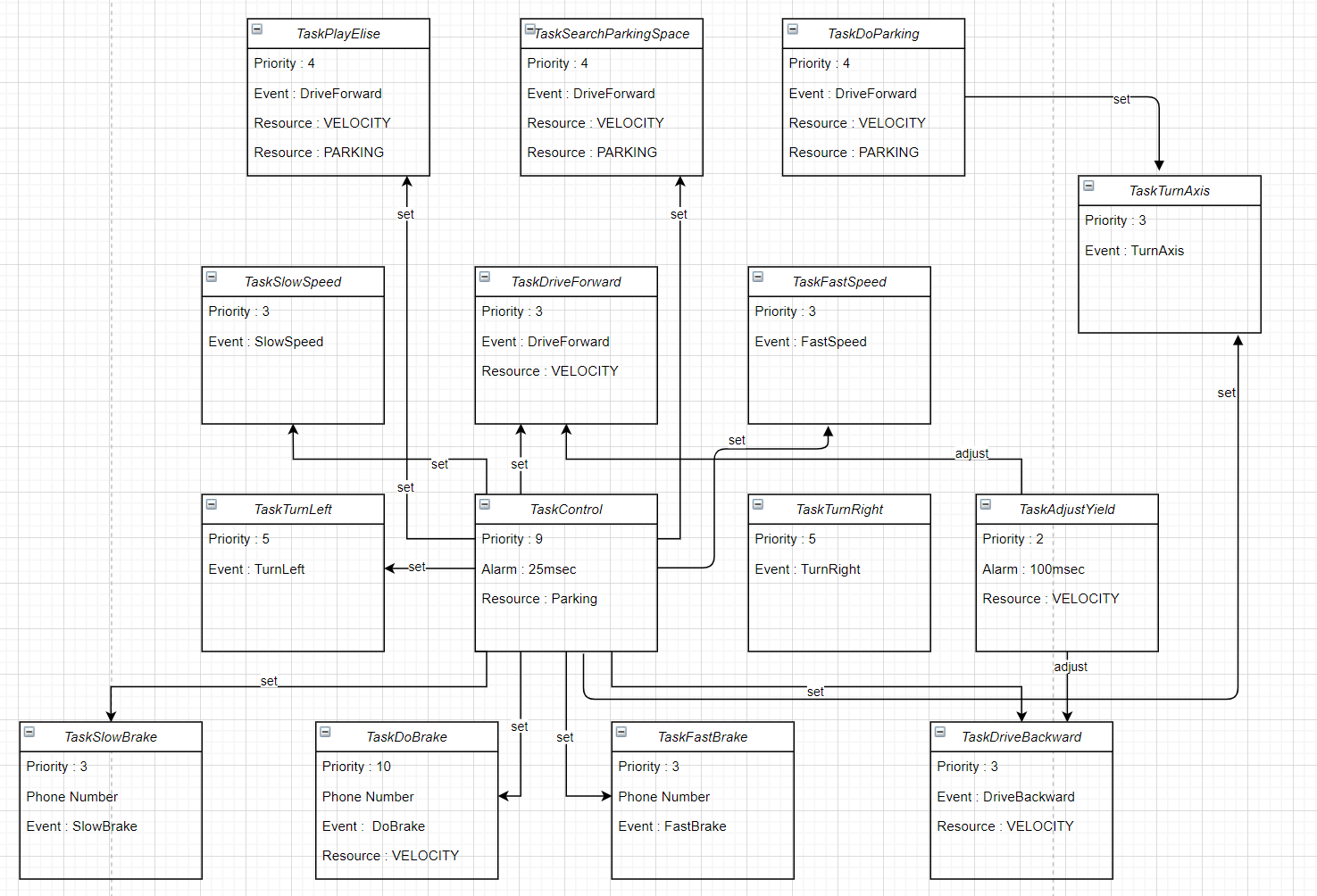
차량 2에서 목표 삼아 중점적으로 설계했던 부분 역시 부드러운 감/가속이 있었고 차량 2가 차량 1의 동선을 원활히 추적할 수 있도록 목표하였다. 이 경우 차량 1과 차량 2의 움직임에 대한 수학적 고찰이 필요했다.

차량 1과 차량 2의 구동 시 동작은 [그림 1]과 같은 구조로 이루어진다.



[그림 1] 프로젝트 구조 개요

# **2. 차량 1**



[그림 2] 차량 1 전체 TASK 구조

# **2-1. 주 기능**

각 기능들은 Task로 만들어서 Event가 set 됐을때 실행하도록 만들었다. TaskControl 에서 버퍼에 입력값(핸드폰 조작)이 들어오면 해당하는 기능을 가진 Task의 Event를 Set 시켜준다.

## **2-1-1. 전 / 후진**

MAX\_SPEED 라는 변수로 최대 속도를 설정해서 속도가 MAX\_SPEED되기 전까지 증가하는 방법으로 구현했다. 육안으로 속도가 서서히 증가하는것을 볼 수 있다.

전진은 후진하다가 전진, 전진으로 2가지 STATE로 나눴고 후진은 전진하다가 후진, 후진으로 2가지 STATE로 나눴다.

## **2-1-2. 좌/우 조향**

좌측 회전시 중심축을 기준으로 최대 회전 반경 전까지 회전을 한다. 동일하게 우측 회전시 중심축을 기준으로 최대 회전 반경 전까지 회전을 한다.

좌우 회전 종료시 회전한 반경이 중심축과 우리가 지정한 일정범위(NEUTRAL\_ZONE) 기준으로 더 틀어져 있으면 이 범위안에 들어올수 있게한다. 실제 자동차의 스티어링 휠같은 기능이다.

## **2-1-3. 속도, 브레이크 모드 변환**

고속/저속 변환은 MAX\_SPEED 값을 변화시켜서 조정이 가능하다.

브레이크 모드도 마찬가지로 브레이크 변수인 BRAKE\_RATIO 값을 변화시켜서 급/천천히 브레이크 모드가 가능하다.

## **2-1-4. 브레이크**

중요한 기능인 만큼 우선순위를 높게 주었다. 어떠한 브레이크 모드를 설정하든지 BRAKE\_RATIO 값에 따라 자동적으로 모드에 맞게 브레이크가 작동한다.

# **2-2. 보조 기능**

## **2-2-1. 좌/우 조향 최대 각 측정 및 정면 축 인식**

처음 시스템이 시작되고 가장 우선순위가 높은 TaskInitialize 가 실행된다. 이 초기화 과정에서

좌우 최대 회전 반경을 getCount로 계산한다. 두개를 합한것에 2를 나눈것을 중심축으로하고 두 값의 차를 2로 나누고 거기에 30을 뺀 값을 최대 회전 반경으로 설정한다. 30을 뺀 이유는 너무 많이 바퀴가 돌아가면 안되서 우리가 적당히 줄인것이다.

## **2-2-2. 후륜 모터 수율 제어**

양쪽 모터의 수율이 달라서 같은 시간동안 getCount() 값을 측정해보면 다른 값이 나왔다. 우리는 이 값의 차이 최대한 줄이고자 100msec 주기의 TaskAdjustYield를 만들어서 양쪽 모터의 각각 이전의 getCount값과 현재의 getCount 값을 차이의 절댓값을 구했다. 현재의 값만 비교하는것보다 이전에서 현재로의 증가량을 비교하는것이 더 정확하다고 생각해서 이런 방법을 써보았다. 단 수율제어는 스티어링휠이 정면을 바라보고 있을때만 작동하도록 했다.

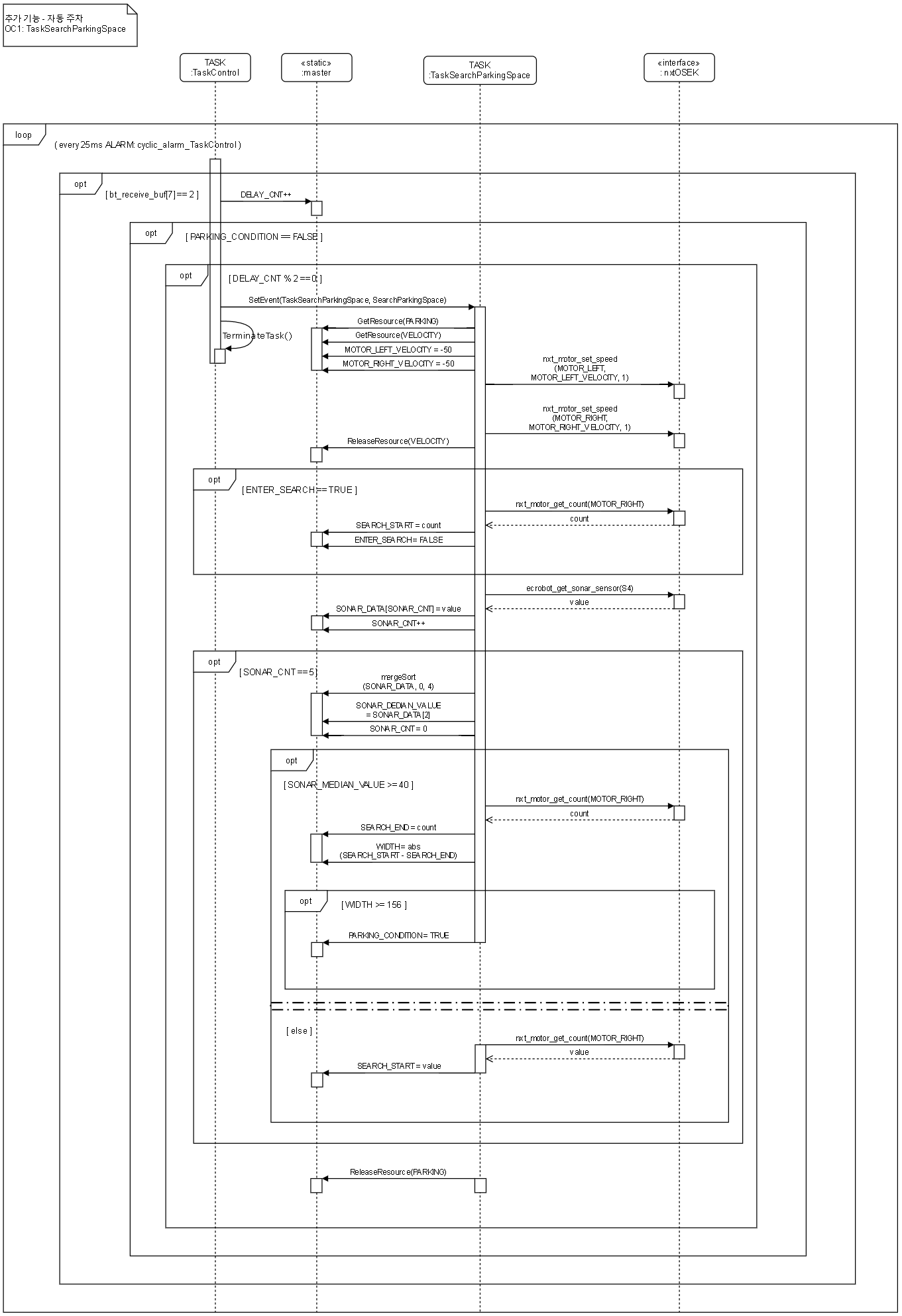
왼쪽 모터에서 구한 절댓값과 오른쪽에서 구한 절댓값을 차이를 구해서 이것을 수율제어의 기준으로 삼았다.

이 차이가 1보다 큰경우 전진인 경우 후륜 좌측 모터가 더 빨리 도는것이므로 해당 모터의 속도를 줄이고, 후진인 후륜 우측 모터가 더 빨리 도는것이므로 후륜 좌측 모터의 속도를 올린다. 이 차이가 -1보다 작은 경우는 그 상황에 알맞게 후륜 좌측 모터의 속도를 조절한다.

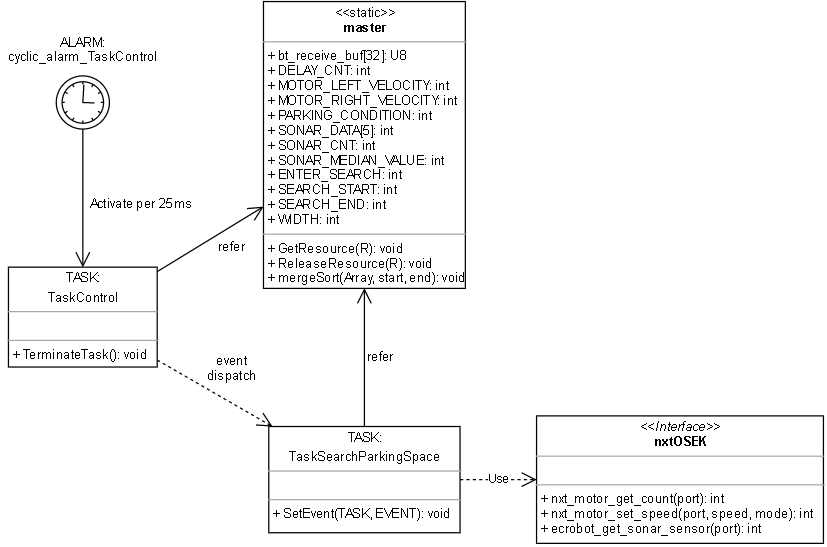
# **2-3. 추가 기능**

## **2-3-1. 자동 주차**

### **2-3-3-1. 주차 공간 탐색(TaskSearchParkingSpace)**



[그림 3] 주차 공간 탐색 Sequence Diagram



[그림 4] 주차 공간 탐색 Design Diagram

추가 기능-자동 주차는 어플리케이션의 B 버튼을 누르면서 활성화할 수 있다. 차량 왼쪽 측면 중앙에 초음파 센서가 있다고 가정한다.

자동 주차를 하기 위해서 먼저 주차하기에 충분한 공간이 있는지 탐색해야 한다. 공간 탐색을 통해 주차 여건이 마련되지 않았다면 전진하며 초음파 센서를 통해 공간을 탐색한다. 초음파 센서는 감지한 값을 NXT에 제대로 전송하기 위해서 최소 50ms의 시간이 필요하다. 25ms마다 실행되는 Event Dispatcher인 TaskControl은 두 번째 실행 때마다 초음파 센서를 활성화하여 거리를 측정한다. 초음파 센서의 측정 값에 이상치가 생길 수 있기 때문에 다섯 칸짜리 배열을 이용해 배열에 초음파 센서의 측정 값을 저장하고, 모두 채워지면 합병 정렬 후 중간 값을 이용해 이상치를 이용할 확률을 낮춘다.

차체의 가로 길이와 세로 길이가 대략 20cm, 30cm이고 초음파 센서가 차체 측면 중간에 있음을 감안해 초음파 측정 값이 40 이상인 채 30cm 이상 전진하는 경우 차체가 주차할 공간이 있다고 판단할 수 있다. 이 때 정확히 30cm 이상 전진했음을 판단하기 위해 원의 반지름과 중심각을 이용한 호의 길이 공식을 사용한다. 호의 길이 공식은 다음과 같다.

**[수식 1] 호의 길이 공식**

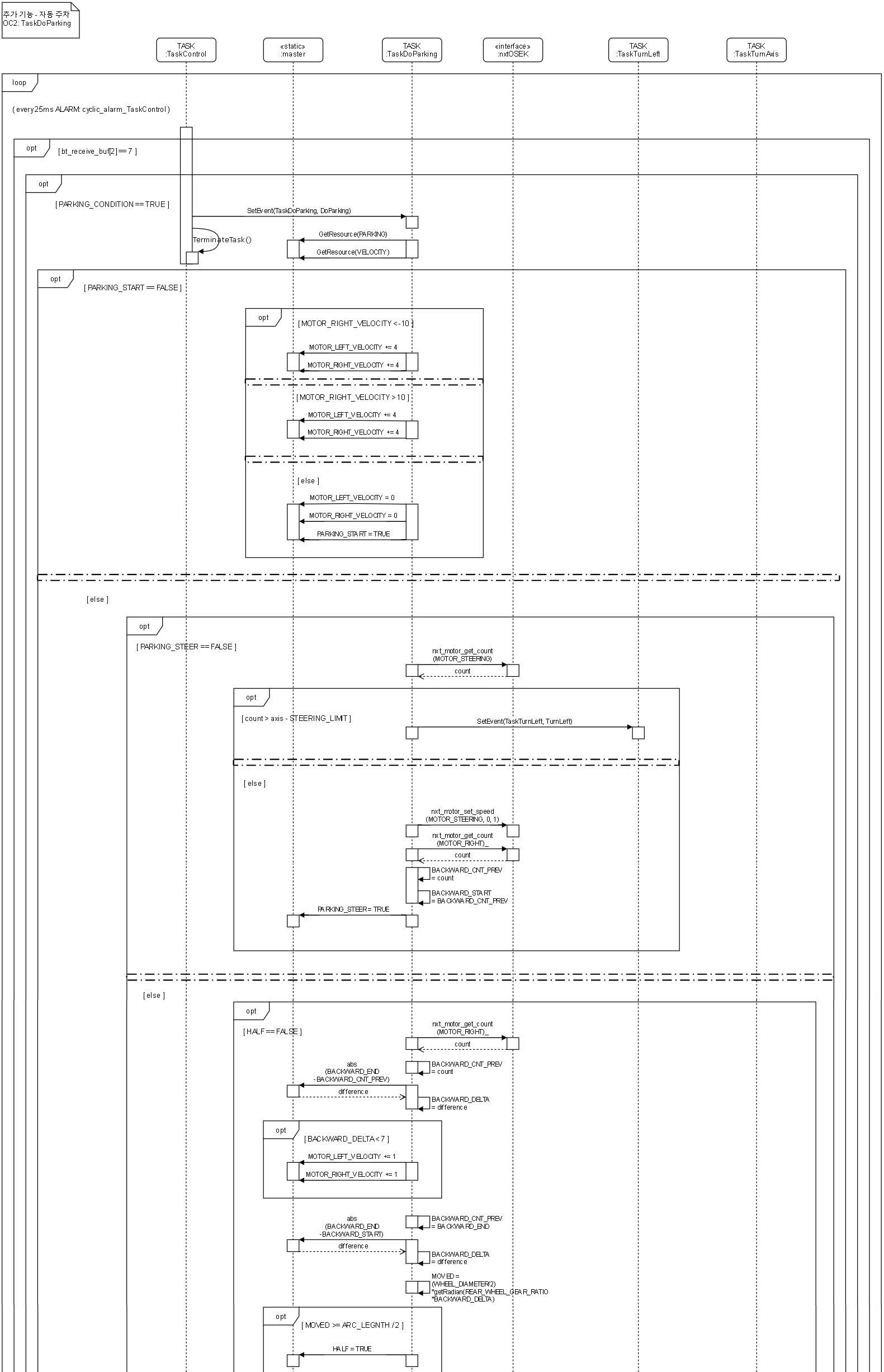
이 때 호의 길이는 차체가 이동한 길이, 반지름은 차량 후륜 바퀴의 반지름, 중심각은 후륜 모터의 회전각 변화량이다. 차량의 후륜 바퀴의 지름은 직접 자를 이용해 측정한 결과 5.5cm이었고 이를 통해 반지름을 알 수 있다. 후륜 모터의 회전각 변화량은 nxt\_motor\_get\_count() API 이용해 실시간으로 측정한다.

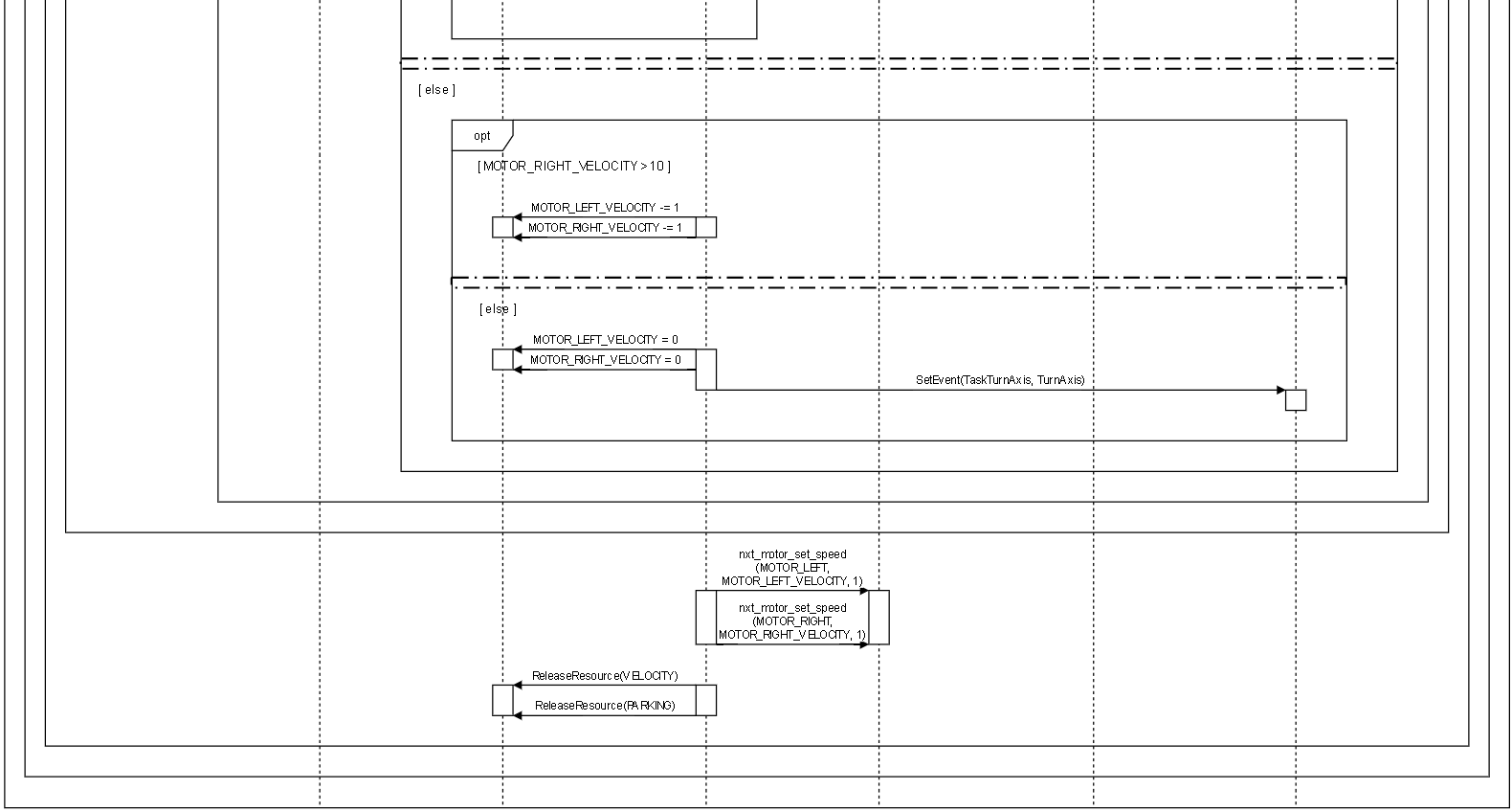
한 편 차체를 살펴본 바 후륜 모터의 바퀴 회전 기어비는 1 : 4였다. 이는 모터가 1 라디안 회전할 때 실제 후륜 바퀴는 4 라디안만큼 회전함을 의미한다. 따라서 위의 공식을 이용하여

라는 수식이 성립하고, 이 경우 모터 회전각 (theta)는 2.73라디안, 약 156도이다. 초음파 센서 값이 40 이상인 채 후륜 모터의 회전각 변화량이 156도 이상이 될 경우 주차 공간이 확보됐다고 판단하여 전역 변수 PARKING\_CONDITION을 TRUE로 세팅한다. 초음파 센서 값이 40 미만이라면 후륜 모터의 회전각 변화량이 0부터 다시 증가하게끔 구현한다.

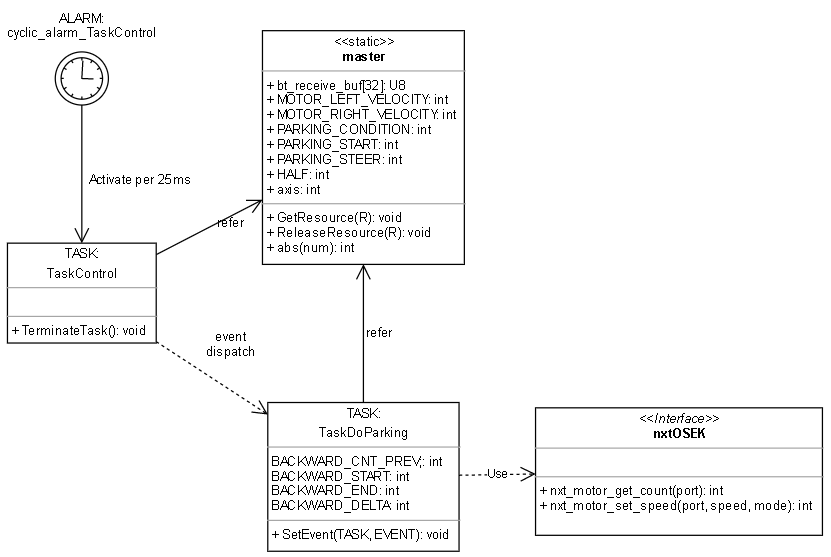
**2-3-3-2. 주차**

(다음 페이지부터)





[그림 5] 주차 Sequence Diagram



[그림 6] 주차 Design Diagram

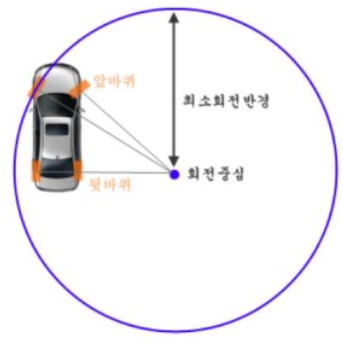
주차 공간이 확보됐다면 주차를 실시한다. 차량이 주차 공간을 지나친 다음 후진 방향으로 적절히 선회하여 주차 공간에 정확히 주차하는 것을 목표로 한다.

주차와 관련된 Task는 TaskDoParking이며, 해당 Task의 단계를 나누어 보자면 다음과 같다.

1. 차량 정지
2. 스티어링 휠 조작
3. 후진 가속과 감속

가장 먼저 차량을 정지시켜야 한다. 차량의 속도를 천천히 늦추어 완전히 정지시킨다. 다음으로 주차할 방향에 따라 스티어링 휠을 조작한다. 초음파 센서가 차량 왼쪽 측면에 있다고 가정하였고 이에 따라 주차 방향은 차량 좌측 하단이다. 따라서 스티어링 휠을 좌향 최대 각으로 조정될 때까지 휠을 회전시킨다. 좌향 최대 각까지 회전됐다면 전역 변수 PARKING\_STEER를 TRUE로 세팅함으로써 2번 단계까지 완료됐음을 판단한다. 이 때 후륜 모터의 회전각을 저장해놓는다.

2번까지 완료됐다면 다음으로 후진을 진행한다. 후진을 진행할 때 정확히 주차 공간에 차량을 주차시키기 위해서는 차체의 최소회전반경을 알아야 한다.



[그림 7] 최소 회전 반경

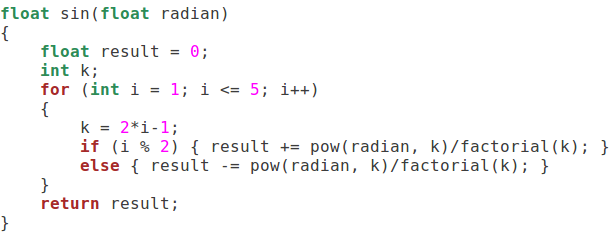
최소회전반경이란 차량이 최대 조향 후 전/후진 시 그릴 수 있는 가장 작은 원의 반지름이다. 최소회전반경을 구하는 공식은 다음과 같다.

**[수식 2] 최소회전반경 공식**

수식과 같이 축 간 거리와 스티어링 휠의 최대 조향 각을 알 수 있으면 최소회전반경을 구할 수 있다. 축 간 거리란 차량 앞바퀴 축과 뒷바퀴 축 사이의 거리를 말한다. 차량 1의 축 간 거리를 직접 자를 통해 측정한 결과 약 18cm이었다. 최대 조향 각은 차량 1 구동 시 좌/우 최대 조향 각을 구하므로 이를 이용한다.

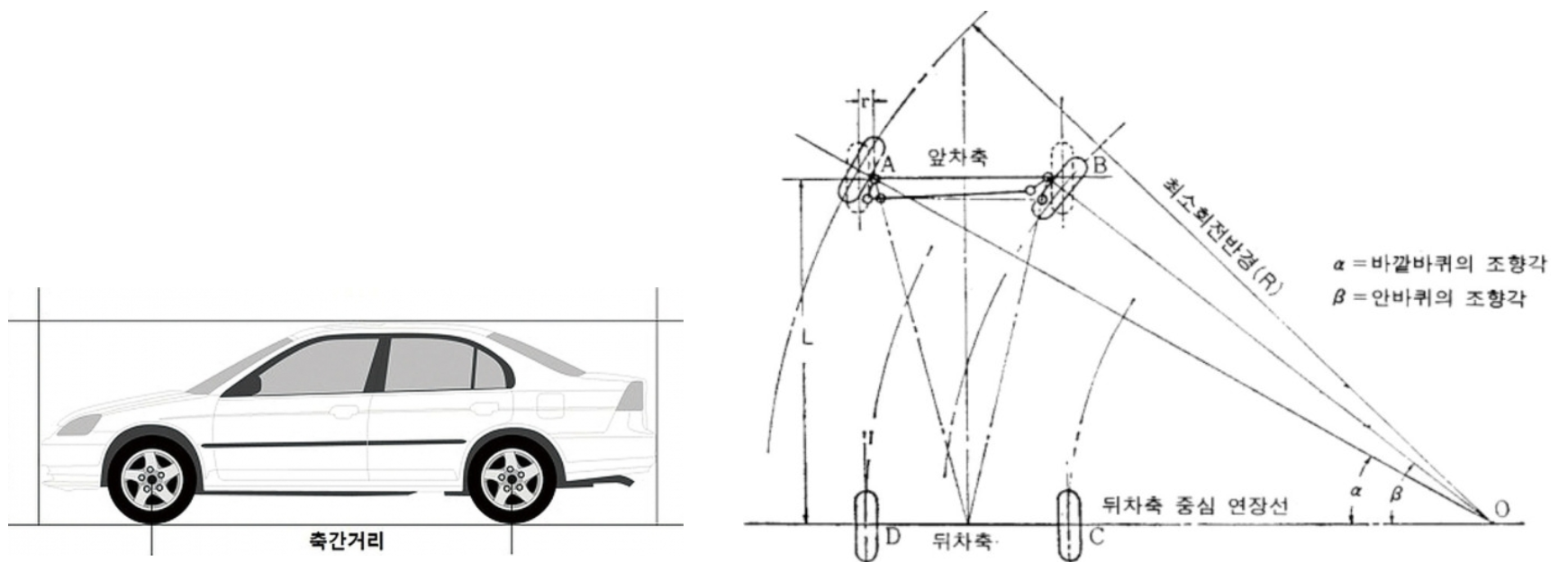
좌/우 최대 조향 각을 구하는 TaskInitialize에서 최소회전반경을 구해놓는다. 최소회전반경을 계산할 때 삼각함수 sin을 사용해야 하는데, sin은 C 라이브러리의 math 파일을 이용해 사용할 수 있지만 코드 용량을 줄이기 위해 직접 구현하여 사용했다. sin을 코드로 구현하기 위해서는 테일러 급수 성질을 이용해야 한다.

**[수식 3] 삼각함수 sin을 테일러 급수로 표현한 수식**



[그림 8] 삼각함수 sin 구현 코드

테일러 급수 전개 시 다섯 번째 항까지만 전개하도록 구현하여 오차를 허용하되 계산 속도를 높혔다.



[그림 9] 축 간 거리와 앞 바퀴 조향각 보충 그림

최소회전반경을 알았다면 호의 길이 공식을 통해 차량이 최소회전반경을 가지는 원 위에서 90도만큼 후진 방향으로 회전한 거리를 구할 수 있다. 해당하는 거리를 ARC\_LENGTH라 하자.

ARC\_LEGNTH 또한 차량 1 구동 시 TaskInitialize를 통해 구해놓는다.

TaskDoParking에서 2번 단계까지 완료되었다면 차량이 정지한 채 스티어링 휠은 좌측 조향 최대 각을 유지하는 상태이다. 이 상태에서 후진을 하여 이동한 거리가 ARC\_LENGTH만큼 정확히 움직인다면 차량은 최소회전반경을 가지는 원 위를 90도만큼 후진 방향으로 회전한 것이다. 그리고 이는 좌측 후방에 위치한 주차 공간에 완벽한 주차를 수행한 것이다.

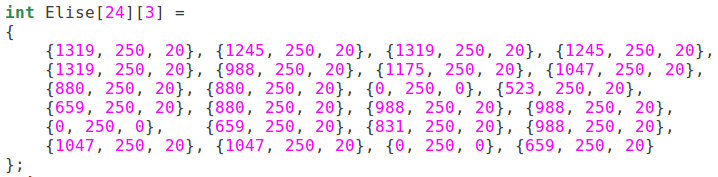
3번 단계를 수행할 때 후진을 매끄럽게 진행하기 위해 가속과 감속을 적절히 넣었다. 앞서 구한 ARC\_LENGTH의 절반만큼 이동할 때까지 가속하며 이후엔 감속한다. 가속은 속도가 무한정 높아지지 않도록 후륜 모터가 적절한 회전 속도를 가질 때까지만 모터의 속도를 증가시킨다. 차량이 초당 15cm 정도 움직인다면 적절한 속도를 가진다고 생각했다. TaskControl이 수행되는 주기인 25ms마다 몇 도만큼 후륜 모터가 회전해야 하는지 수식을 세워보면 다음과 같다.

계산 결과 (세타)는 약 7.81보다 높아야 한다. 따라서 후륜 모터의 회전각 변화량이 7보다 클 때까지 속력을 증가시킨다.

3번 단계 후진까지 완료되었다면 자동 주차는 완료되었다. 이후 정지한 채 좌측 최대 조향 각으로 회전되어 있던 앞 바퀴를 정향으로 되돌려 놓는다.

### **2-3-3-2. 음악 재생**

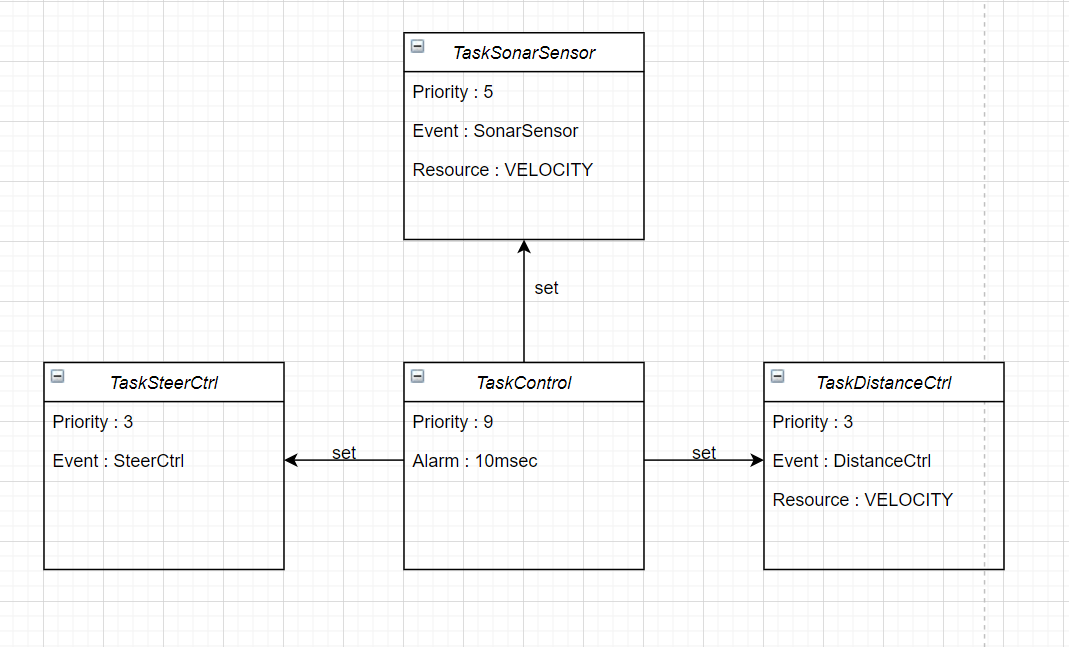
2-3-3-1. 주차 공간 탐색을 마치고 2-3-3-2. 주차를 실시할 때 동시에 주차를 실시하고 있음을 직관적으로 인지할 수 있도록 음악 재생 기능을 구현했다. 재생할 음악은 클래식 음악 ‘엘리제를 위하여’ 이다. 3/8박자로 진행되는 해당 곡의 악보를 참고하여 24x3 배열에 음과 길이, 출력을 담았다.



[그림 10] 엘리제를 위하여 악보가 담겨있는 배열

한 편 TaskControl은 25ms마다 알람으로 실행되고, 음악 재생 기능을 수행하는 TaskPlayElise를 3/8박자 악보에 맞게 연주되도록 250ms마다 실행되어야 한다. 따라서 TaskControl이 TaskPlayElise에게 event dispatch를 250ms마다 수행할 수 있도록 DELAY\_CNT 변수와 modulo(%)를 이용하여 구현하였다.

# **3. 차량 2**



[그림 11] 차량 2 전체 TASK 구조

차량 2는 2개의 Sonar Sensor를 이용하여 각각의 거리를 측정한다. 이를 통해 앞 차량과의 거리를 계산한 내용을 토대로 Motor Actuator(B, C)를 제어하여 전진, 브레이크 그리고 후진을 결정한다. 또한 두 Sonar Sensor값의 차이를 이용하여 앞차 코너링에 대한 정보를 연산하여, 앞차의 선회 방향과 그 각도를 판단하여 Motor Actuator(A)를 제어한다.

# **3-1. 주기능**

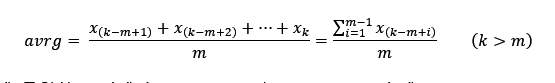
TaskSonarSensor는 50ms마다 Sonar Sensor의 값(0~255)를 받는다. Sonar Sensor의 값은 알 수 없는 원인으로 인해 큰 오차가 생기는 값을 전달한다. 그렇기 때문에, Sensor Filtering이 필수적이다. 대표적인 Filtering Algorithm으로 Min/Max Filter와 Median Filter가 있는데, Circular Buffer를 적용한 Min/Max Filter와 Median Filter의 특성은 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  | Min/Max Filter with buffer | Median Filter |
| react time | 50ms | 50ms \* (size of buffer) |
| adapt to change | flexible | not flexible |
| improve accuracy | use average value of buffer | increase size of buffer |
| Reliability | approximate | approximate |
| error detection | o | - |

**[표 1] Min/Max Filter, Median Filter**

데이터의 정확성을 높이기 위해서 Median Filter의 경우에는 buffer의 크기를 늘리는 수밖에 없다. 그렇게 하면 buffer 크기의 곱만큼 반응 속도가 느려 진다. 하지만 Min/Max filter의 경우에는 circular buffer를 적용하고 buffer값의 평균을 50ms마다 구하는 방식으로 전자의 경우보다 거리의 변화에 빠르고 유연하게 반응한다.

만약 x1,x2,x3...xn의 실제 거리가 오차 없이 센서에 전달되고,개의 buffer를 갖는 Min/Max filter의 경우에는 후에 측정될 average는



가 된다. 반면에 동일한 조건에서 Median filter는 k\*m\*50ms후에



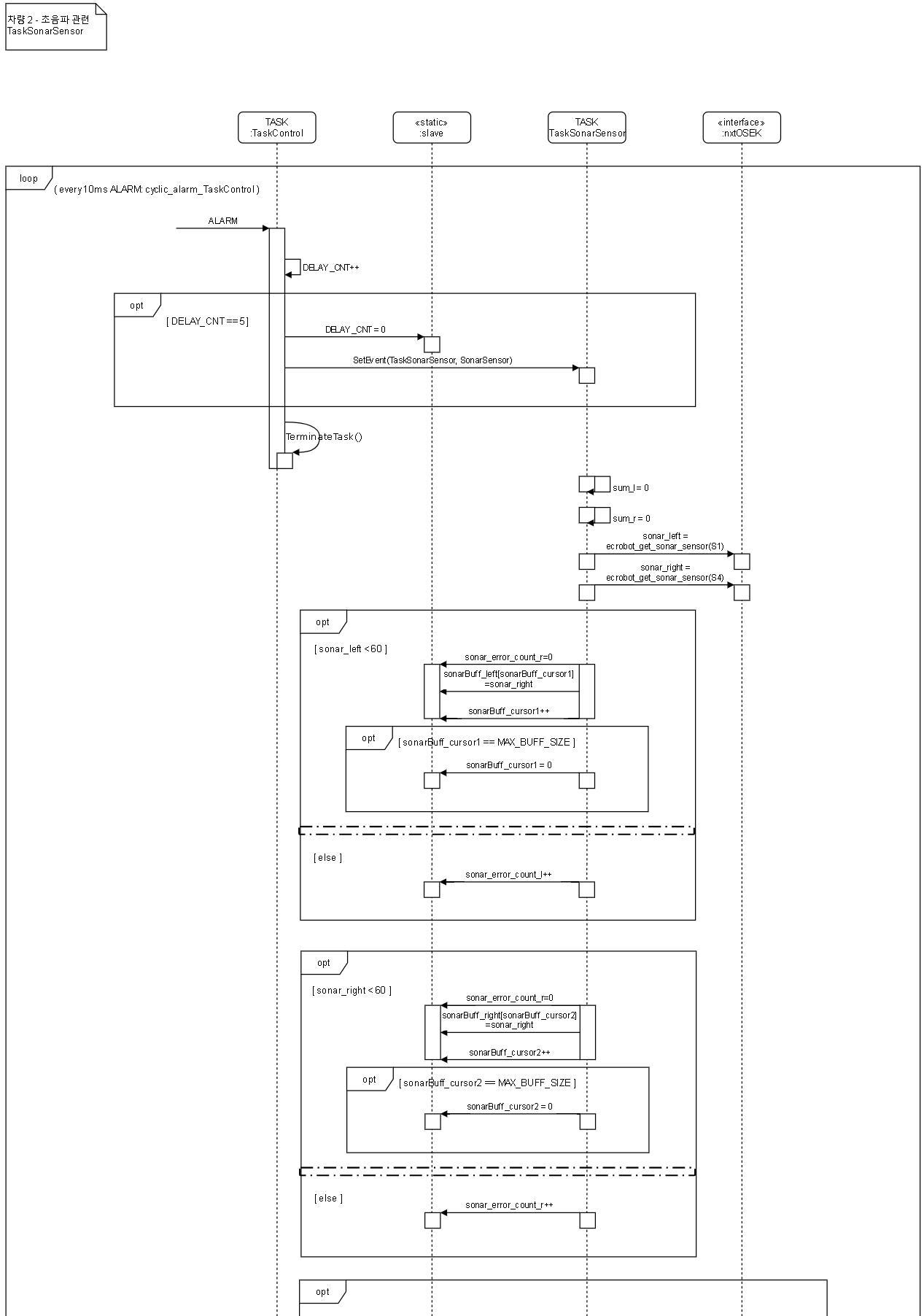
를 거리라고 판단한다. 만약 실제 거리x1,x2,x3...xn가 Linear하게 변화한다면 현재 실제 속도와 측정값의 오차는 다음과 같이 동일하게 된다.

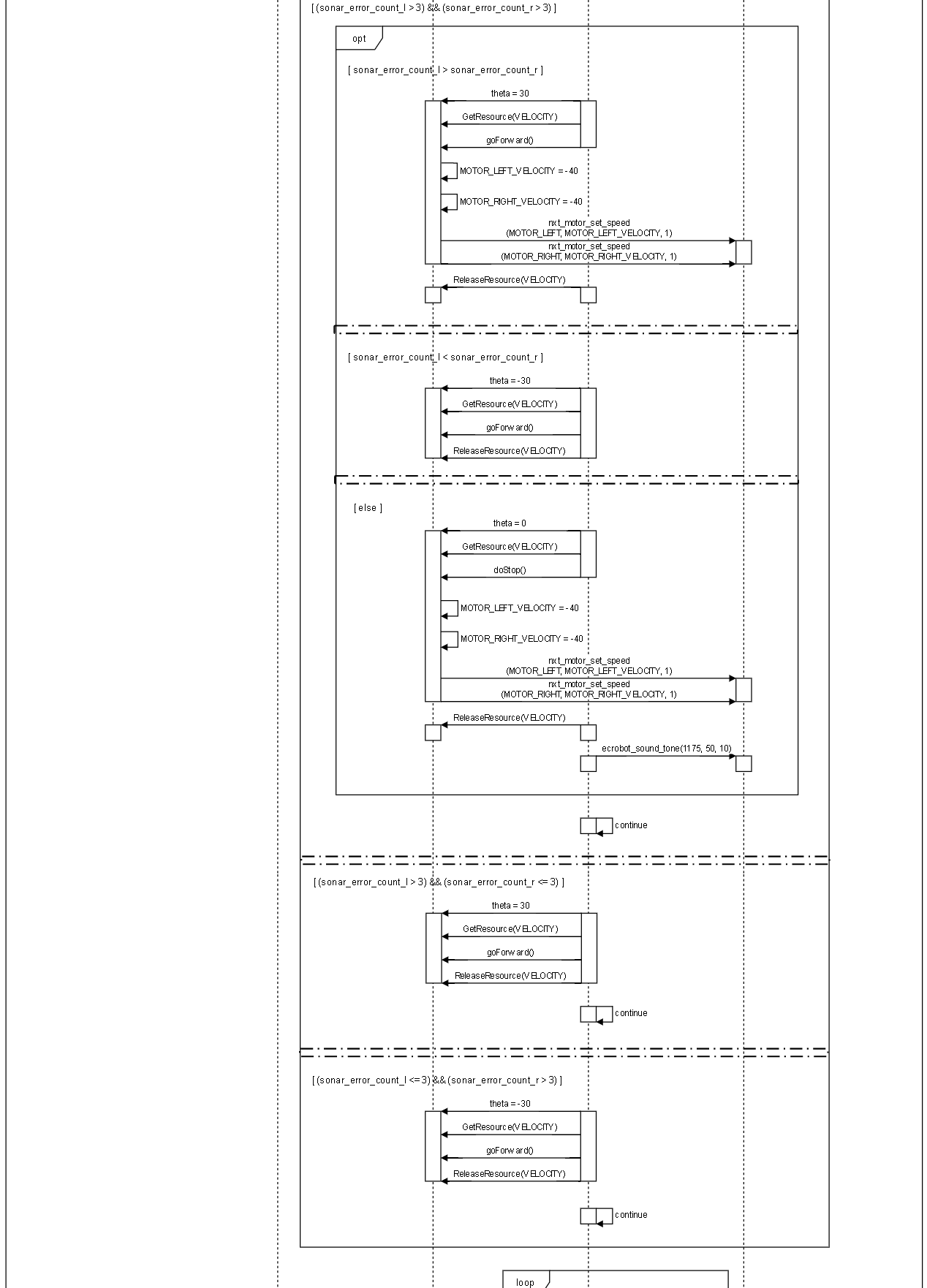


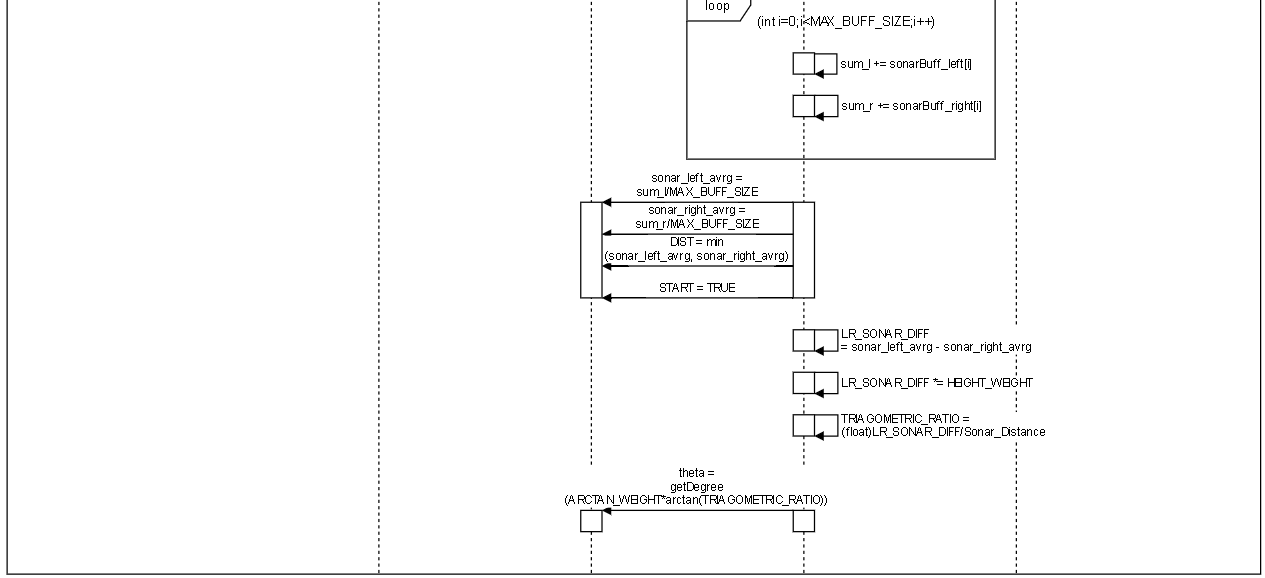
하지만 Median filter의 경우에는 Min/Max filter가 m번 측정될 동안 1번의 계산을 수행하기 때문에 오차가 다르지 않으면서 react time면에서 손해를 보는 셈이다. 또한, 이 오차는 실제 거리가 더욱 급격하게 변하면 오히려 Min/Max filter의 경우에 줄어든다.

결정적으로 circular buffer를 적용한 Min/Max filter를 사용하게 되면, min/max에서 벗어나는 값들을 조건문에서 error로 정의하고, 이를 다른 기능에서 활용하기에 용이하다.

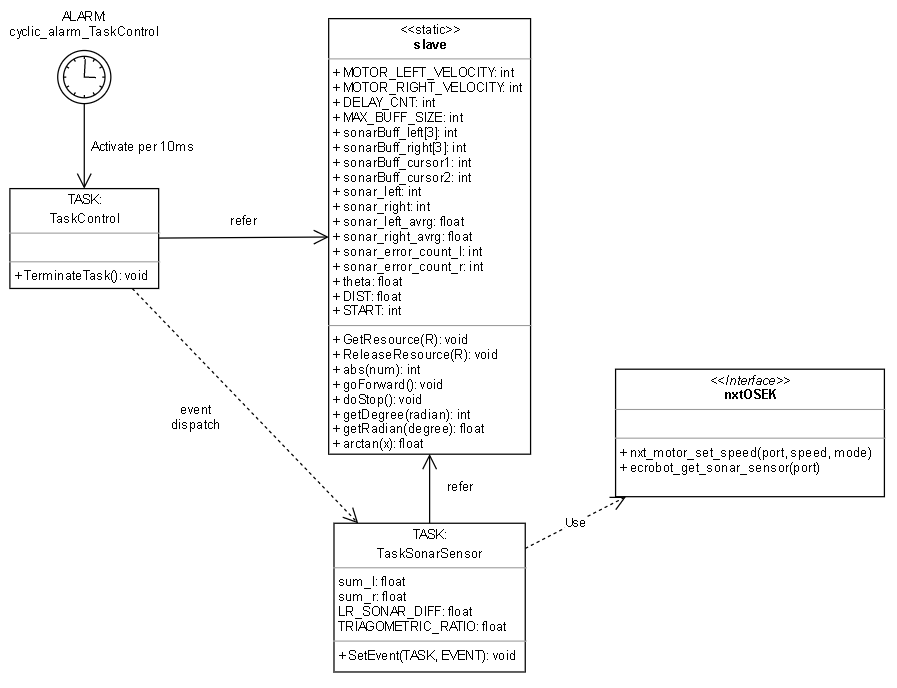
위와 같은 이유로 본 프로젝트에서 차량 2의 Sonar Sensor filtering algorithm은 circular buffer를 적용한 Min/Max filter를 채택하였다.





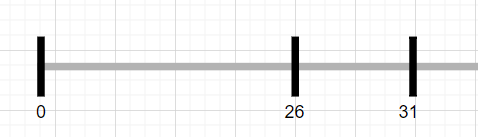


[그림 12] 차량 2의 초음파 알고리즘을 이용하는 TaskSonarSensor의 Sequence Diagram



[그림 13] TaskSonarSensor의 Design Diagram

## **3-1-1 전/후진**



안전거리 : 26

적절한거리 : 26 ~ 31

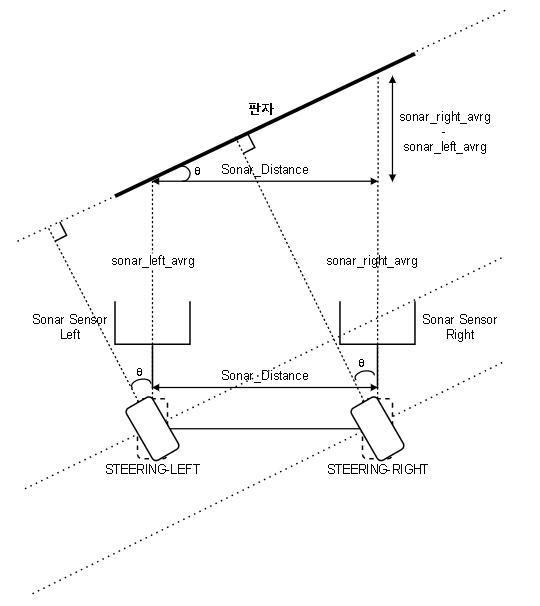
전/후진, 고속/저속 일때 따라가는 것은 TaskDistanceCtrl 에서 수행한다. 우리는 앞차와의 거리를 안전거리, 적절한 거리 2가지로 나누었다.

안전거리 26 보다 가까운 경우 감속하면서 거리가 가까워지면 정지, 후진한다. 적절한 거리인 26~31일 경우DIST\_DELTA(현재 거리- 이전거리) 를 활용해서 해당 거리를 유지할 수 있도록 속도를 지속적으로 조절한다.

31 이상 멀어지면 거리가 멀다고 판단한다. 거리가 많이 멀수록 큰 가속계수를 사용하여 거리를 좁히도록 구현한다. 적절한 거리 이내가 되면 다시 전술했던 내용처럼 해당 거리를 유지할 수 있도록 한다.

## **3-1-2. 좌 / 우 조향**

차량 1이 방향을 변경하여 주행하는 경우 차량 2가 차량 1을 효과적으로 따라갈 수 있도록 조향 기능을 사용해야 한다. 조향 기능을 구현할 때 차량 2의 두 초음파 센서 간 거리 Sonar\_Distance와 두 초음파 센서 측정 값의 차이를 이용해 차량 1의 조향 각 θ를 구할 수 있다.



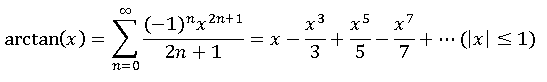
[그림 14] 차량의 조향 각 θ 유도 모식도

차량 1의 조향 각 θ를 이용한 삼각 함수로 tangent를 사용하면 다음과 같이 수식이 성립한다.

따라서 조향 각 θ를 구하기 위해서 arctan를 사용한다.

수식의 우변 분자가 sonar\_right\_avrg - sonar\_left\_avrg가 아닌 이유는 스티어링 휠 모터의 정향 축 기준으로 좌향 시 음의 회전 각을 가지기 때문이다. [그림 14]를 보면 두 초음파 측정 값의 차이, 즉 삼각형의 높이가 양수인 경우 차량 2는 조향 각을 좌측으로 판단한다. 이 경우 수식의 우변 분자가 음수여야 조향 각 θ가 음수가 된다.

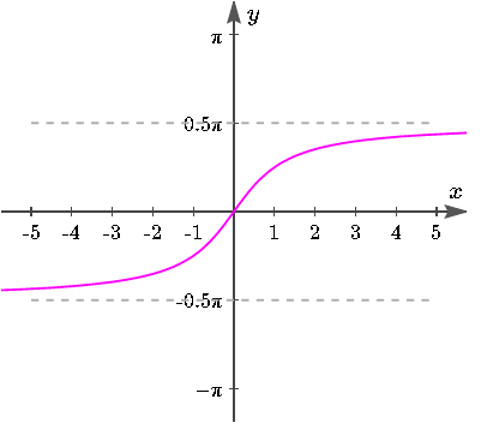
arctan는 c 라이브러리의 math 파일을 이용해 사용할 수 있지만 코드 용량의 효율을 위해 직접 구현하여 사용했다. arctan를 코드로 구현하기 위해서는 테일러 급수 성질을 이용해야 한다.



**[수식 3] 역삼각함수 arctan를 테일러 급수로 표현한 수식**

arctan을 구현할 때 주의할 점은 arctan의 인자로 들어오는 x의 값이 -1 이하거나 1 이상인 경우 각 분해를 통해 오차를 줄여야 한다는 점이다. 역삼각함수 arctan을 테일러 급수로 유도할 때 정의역이 해당 범위로 주어져 그 외의 값이 들어오는 경우 오차가 기하급수적으로 증가한다.

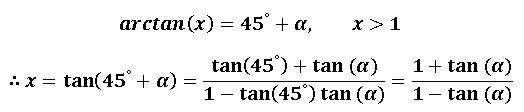
이 경우 삼각함수의 각 분해 성질을 이용하면 문제를 해결할 수 있다.



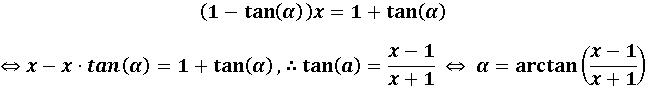
[그림 15] arctan 그래프

**[수식 4] 삼각함수 tan의 각 분해 공식**

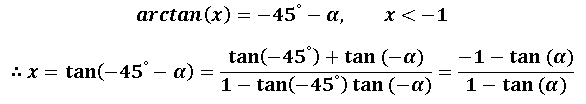
[그림 15]에서 확인할 수 있듯이 arctan(x)에서 x가 1 이상이면 그 값은 45도 이상이고, x가 -1 이하이면 그 값은 -45도 이하이다. 이 때 [수식 4]의 각 분해 공식을 이용하면



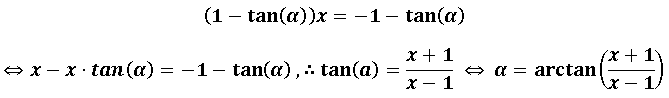
와 같이 유도할 수 있는데, 이 때 x는 주어진 값이므로



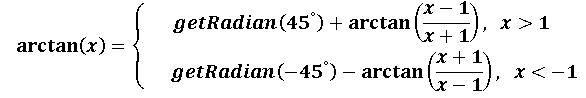
이다. 반대로 x가 -1 이하인 경우,



와 같이 유도할 수 있고, 마찬가지로 x는 주어진 값이므로

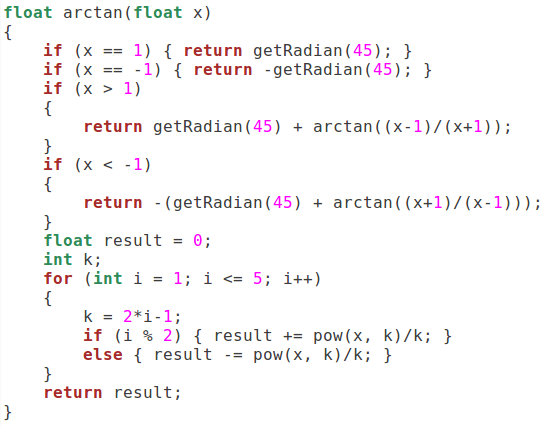


가 된다. 결론적으로



임을 알 수 있다.

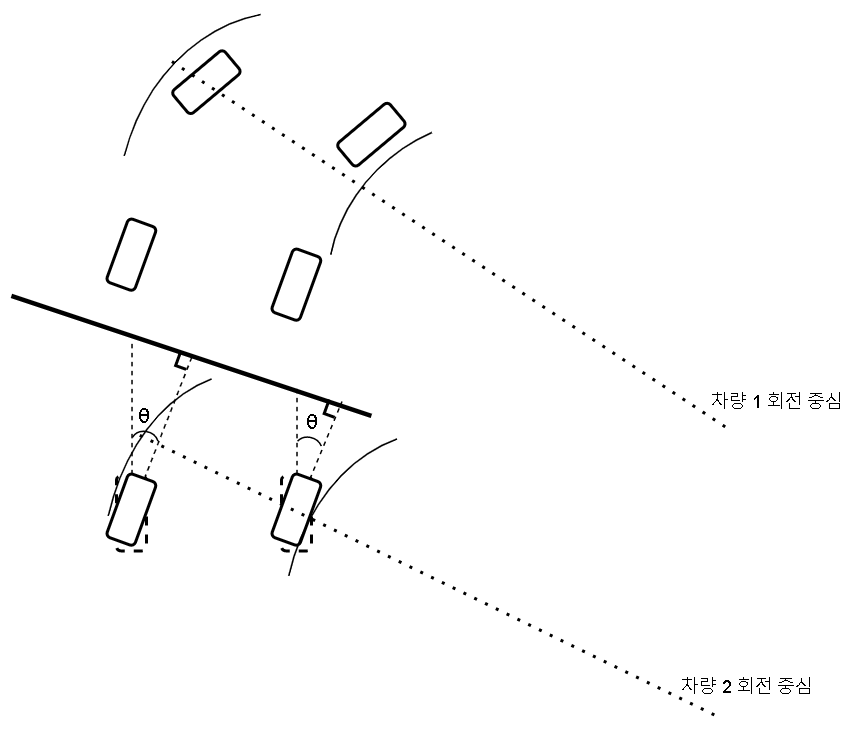
이를 코드로 [그림 16]와 같이 구현할 수 있다.



[그림 16] 역삼각함수 arctan 구현 코드

초음파 측정 값을 이용하기 때문에 조향 각 θ를 계산하는 것은 TaskSonarSensor에 구현되어 있다. 이 θ값을 이용해 TaskSteerCtrl에서 차량 2의 조향을 조작한다. 차량 1과 마찬가지로 차량 2에서도 구동 시 좌/우 최대 조향 각을 측정하고 정향 축을 인지하는데, 이 축을 기준 삼아 스티어링 휠이 정확히 θ만큼 회전하도록 구현한다.

한 편, 차량 1과 차량 2의 주행 속도나 초음파 센서 값 측정의 지연때문에 차량 2의 조향 각을 차량 1의 회전 각으로 설정하면 차량의 회전 궤도가 잘 들어맞지 않는다.



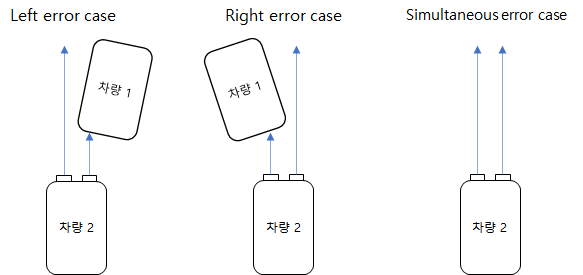
[그림 17] 차량 1과 차량 2의 회전 모의도

따라서 측정한 (세타) 값보다 좀 더 완만한 각을 이용하여 조향을 조작하는 것이 바람직하다. 이를 위해 1보다 작은 값 ARCTAN\_WEIGHT를 측정된 (세타)값에 곱해주어 실제 조향 각으로 사용하였다.

한 편 차량 1 동선 추적을 원활히 하기 위해 Error count 알고리즘을 사용한다. Error count 알고리즘에 대한 설명은 다음과 같다.

Min/Max Filter에서 원하는 범위를 벗어난 값을 센서에서 받아왔을 경우에, error count를 증가시킨다. 이때, 좌측 sonar sensor와 우측 sonar sensor의 error count를 각각 기록한다.

허용 범위의 센서값이 들어왔을 경우에는 이 error count를 0으로 reset한다.



[그림 18] Error count 알고리즘을 이용한 모의도

만약 차량 1이 급격하게 우회전을 하다가 sensor가 이를 놓치게 되면 (Left error case) 좌측 sonar sensor의 error count가 더 높은 값을 갖게 될 것이고, 앞 차량이 오른쪽 어딘가에 있다고 가정하고 우회전하여 탐색한다. 반대로 차량 1이 좌회전 중에 놓치게 되면 (Right error case) 우측 sonar sensor의 error count가 더 높은 값을 갖게 된다. 그렇다면 좌회전하여 탐색한다. 만약 어떤 이유로 인해 차량 1이 사라지게 되면 (Simultaneous error case) 양쪽의 error count가 같은 값으로 증가하게 될것이다. 이때는 차량을 급정지하고 경고음을 울리게 했다.

# **5.** **맺음말**

지금까지 했던 프로젝트들 중에서 가장 흥미로웠고 흥미만큼이나 난이도가 상당했다. 실제 자동차 제어장치에 들어가는 OESK OS를 이용해 프로그래밍하며 초심자라면 단순한 기능 하나조차 여러 기능들 사이에서 제대로 동작시키기 위해 지고의 노력을 경주해야 함을 느꼈다. 특히 센서에 관련해서 차량 2의 두 초음파 센서가 서로 간섭을 일으켜 코드를 제대로 작성하여도 원하는 대로 동작하지 않는 경우 등 하드웨어와 관련된 문제에도 골머리를 앓았다.

자동차 시장에서 소프트웨어 인력이 간단한 노력으로 만들어지지 않음을 이번 프로젝트를 통해 몸소 느낄 수 있었다. 이번 기회를 통해 앞으로 겪게 될 실시간 운영체제 시스템 설계 부문에 있어 두 단계는 성장했음을 느낀다. 겪었던 여러 시행착오만큼이나 깊은 성취감을 느낀다