집중교육 OSEK 프로젝트

귀여운 고슴도치

201420988 강현구
201421002 한수찬
201620486 오승민
201620950 강한결

목차

그림 목차

1 개요

1.1 프로젝트 목표

블루투스를 이용해 조종할 수 있는 자동차(Control Car)를 만들고 그 뒤를 적절한 거리를 유지하며 따라갈 수 있는 자동차(Follow Car)를 구현하는 것이 이번 프로젝트의 목표다. Control Car는 블루투스 연결을 통해 전진, 후진, 좌회전, 우회전, S회전, Circle회전, 저속주행, 고속주행, 즉시 브레이크, 천천히 브레이크, 되돌아가기의 기능들을 수행해야 하며, Follow Car는 앞차의 직진, 후진, 회전 등의 다양한 상황에서 부딪히지 않고 지속적으로 안전 거리를 유지하며 따라갈 수 있어야 한다.

1.2 프로젝트 배경

자동차는 NXT LEGO를 이용하여 만든 차량이다. 각 차량 내부의 OS는 OSEK OS를 사용하였다. 각 차량의 하드웨어 조작을 위해서 NXT API in C를 이용했다. 각 차량을 조작하는 OSEK OS가 제공하는 interface를 사용한다.

2 요구사항

2.1 Control Car

- 직진 / 후진 기능 요구 사항

블루투스 통신을 통해서 입력이 들어오면 뒷바퀴의 회전을 통해서 자동차를 앞/뒤로 움직일 수 있어야 한다. 뒷바퀴를 제어하기 위해서 NXT 모터의 속도를 조작할 수 있어야 한다.

- 차량 조향 기능 요구 사항

블루투스 통신을 통해서 입력이 들어오면 앞바퀴의 회전을 이용해서 차체를 좌/우로 조작하는 차량의 조향을 해야 한다. 입력이 끝나면 앞바퀴를 원래 상태로 되돌려야 한다. 차량의 조향을 위해서 앞바퀴를 일정 각도 만큼만 회전 하도록 조작할 수 있어야 한다.

- 차량의 브레이크 기능 요구 사항

블루투스 통신을 통해서 입력이 들어오면 자동차의 속도를 급격히 줄일 수 있어야한다. 입력이 들어오지 않는다면 다시 원래 상태로 돌아가야 한다. 차량의 브레이크 기능을 구현하기 위해서 뒷바퀴에 연결되어 있는 motor의 속도를 제어 할 수 있어야 한다.

- 차량의 속도 제어 기능 요구 사항

블루투스 통신을 통해서 입력이 들어오면 자동차의 속도를 고속/저속 모드로 바꿀 수 있어야 한다. 뒷바퀴에 연결되어있는 모터의 power를 제어해야 한다.

2.2 Follow Car

- 앞차가 출발 했을 때 따라서 달릴 수 있는 기능 요구 사항

뒤에 따라가는 차량은 앞차가 출발 했다는 것을 인지 할 수 있어야 한다. 또한 앞차와 너무 멀어지지 않는 시점에 적절히 출발하는 기능이 필요하다.

- 앞차가 속도를 변경 했을 때 맞춰서 속도를 변경할 수 있는 기능 요구 사항

앞차와의 거리를 적절히 유지 해야 한다. 이를 위해서 앞차보다 너무 빨리 달리거나 너무 느리게 달리면 안된다.

- 앞차가 급제동 했을 때 제동하는 기능 요구 사항

앞차가 급제동 했을 때 혹은 천천히 속도를 줄였을 때 뒤에 따라가는 차량은 따라서 멈출 수 있어야 한다.

- 앞차가 방향을 바꿨을 때 따라서 방향을 바꾸는 기능 요구 사항

앞차가 방향을 바꿨을 때 회전을 할 때 뒷차는 앞차의 위치를 잃지 않아야 한다. 이를 위해서 앞차의 방향에 맞춰서 같이 회전해야 한다.

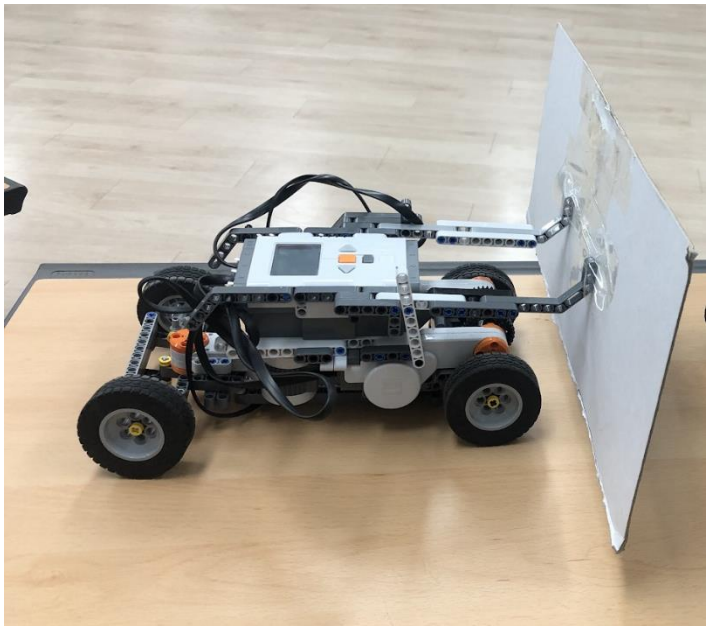
- 앞차가 후진할 때 방향에 맞춰서 후진하는 기능

앞차가 후진한다는 것을 인지하고 이를 위해서 일정 거리 이하에 들어오면 후진하는 기능이 필요하다.

3 시스템 구성 및 설계

3.1 Control Car

3.1.1 H/W



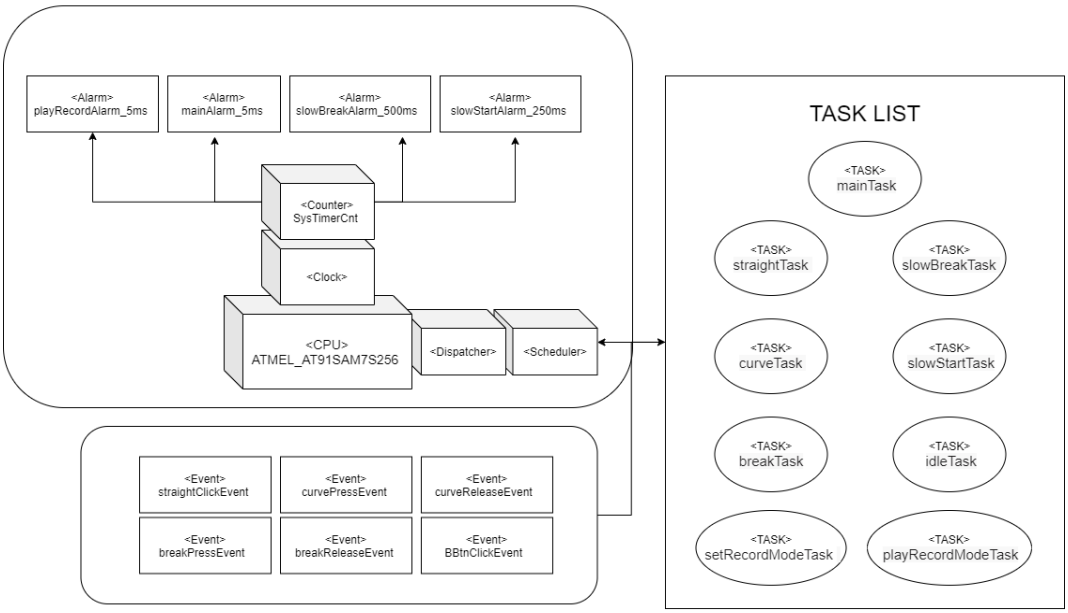
[그림 1] Control Car

- 조향을 하기 위한 조향 모터 1개와 주행을 하기 위한 주행 모터 2개, 모터를 움직이기 위한 본체로 이루어진 차체

- Follow Car 가 Control Car의 운행을 인식할 수 있는 판

3.1.2 S/W

3.1.2.1 OIL Definition



[그림 2] OIL Definition Diagram

3.1.2.2 Task Definition

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	straightTask
PRIORITY	5
TASK TYPE	EXTENDED TASK
IS AUTO START	TRUE
TRIGGER	straightClickEvent
FUNCTIONALITY	차량의 전진 혹은 후진을 할 수 있도록 하는 Task다. straightClickEvent를 사용하여 앞, 혹은 뒤 버튼이 눌렸을 때

	Event를 발생시킨다. state value와 receive buffer의 값을 통해 Port B와 Port C의 스피드를 조절한다. 이 때 record_mode일 경우 record 배열에 차량의 속도와 회전각, 모터가 돌아간 수를 저장한다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	curveTask
PRIORITY	5
TASK TYPE	EXTENDED TASK
IS AUTO START	TRUE
TRIGGER	curvePressEvent, curveReleaseEvent
FUNCTIONALITY	차량의 조향값을 설정하는 Task다. 좌,우 버튼은 눌릴 때와 떨어질 때 값을 다르게 보내므로 두개의 이벤트를 이용해 처리하였다. 버튼이 눌렸을 때 curvePressEvent를 발생시켜 좌, 우에 따라 다른 속도로 Port A의 motor speed를 조정한다. 버튼이 떨어질 땐 curveRelaseEvent를 발생시켜 motor의 카운트가 0 보다 클 땐 speed를 음수로, 0 보다 작을 땐 speed를 양수로 설정해준다. 위와 마찬가지로 record_mode일 경우 record 배열에 차량의 속도와 회전각, 모터가 돌아간 수를 저장한다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	slowBreakTask

PRIORITY	10
TASK TYPE	BASIC TASK
IS AUTO START	TRUE
TRIGGER	slowBreakAlarm_500ms
FUNCTIONALITY	천천히 브레이크를 위한 Task이다. 이 Task는 Alarm으로 500ms 마다 실행되고 있으며 is_slow_break_on이라는 변수가 1 일때만 속도를 10씩 조절하고 모터에 적용한다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	breakTask
PRIORITY	12
TASK TYPE	EXTENDED TASK
IS AUTO START	TRUE
TRIGGER	breakPressEvent, breakReleaseEvent
FUNCTIONALITY	브레이크를 위한 Task다. curveTask와 마찬가지로 버튼이 눌렸을 때와 떼어졌을 때로 나누어 두 가지 이벤트를 Wait하고 눌렸을 때 breakPressEvent를 발생시킨다. breakPressEvent가 발생하면 breakMode로 브레이크 모드를 확인하고 즉시 브레이크모드일 경우 Port B, Port C의 모터 속도를 0으로 설정해주고, 천천히 브레이크 모드일 경우 is_slow_break_on이라는 state 변수를 1로 바꾸어 준다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	setRecordModeTask
PRIORITY	9
TASK TYPE	EXTENDED TASK
IS AUTO START	TRUE
TRIGGER	BBtnClickEvent
FUNCTIONALITY	추가기능을 구현하기 위한 Task다. Task를 통해 record 모드를 시작할 수 있고, record모드에는 차량의 움직임과 속도, 모터의 회전수 등이 배열에 저장된다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	playRecordModeTask
PRIORITY	12
TASK TYPE	BASIC TASK
IS AUTO START	FALSE
TRIGGER	playRecordAlarm_5ms
FUNCTIONALITY	playRecordModeTask는 state 값을 통해 Alarm으로 작동한다. 배열에 저장되어 있는 값들을 역순으로, 반대값으로 차량을 동작시킨다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	slowStartTask
PRIORITY	5
TASK TYPE	BASIC TASK
IS AUTO START	FALSE
TRIGGER	slowStartAlarm_250ms
FUNCTIONALITY	slowStartTask는 차가 전진할 시 천천히 속도를 올리는 Task이다. 이 Task 또한 slowBreakTask와 비슷한 메커니즘으로 Alarm을 통해 250ms마다 실행되고 있으며, startstatus라는 state변수가 1일 때만 speed의 값을 변경하고 모터에 적용한다. startstatus는 straightTask에서 전진일 경우 1로 설정해준다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	mainTask
PRIORITY	9
TASK TYPE	BASIC TASK
IS AUTO START	TRUE
TRIGGER	mainAlarm_5ms
FUNCTIONALITY	데이터값을 확인하기 위해 nxt display에 변수값을 출력하고 receive buffer값에 따라 SetEvent를 해주거나 값을 변경하여 자

	동차에 적용하는 Task이다. Alarm을 통해 5ms마다 실행된다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	idleTask
PRIORITY	1
TASK TYPE	BASIC TASK
IS AUTO START	TRUE
TRIGGER	.
FUNCTIONALITY	Android와 NXT를 블루투스로 연결하기 위한 Task이다.

3.1.2.3 Intro to Core Algorithm

- Steering axis centralization

```
// 조향축을 중앙으로 조절하기 위한 구간
if (nxt_motor_get_count(NXT_PORT_A) != 0 && curve_state == 0) {
  if (nxt_motor_get_count(NXT_PORT_A) > 5) {
    nxt_motor_set_speed(NXT_PORT_A, -45, 1);
  } else if (nxt_motor_get_count(NXT_PORT_A) <= -1) {
    nxt_motor_set_speed(NXT_PORT_A, 45, 1);
  }
}
```

[그림 3] 중앙 조향 코드

모터가 미세하지 못해서 회전 후 되돌아올 때 제대로 정렬되지 않는 경우가 있다. 따라서 제대로 된 조종을 위해서는 조향 축을 중앙으로 조정해줄 필요가 있었고, 5ms 마다 실행되는 mainTask에서 미세하게 값을 계속 조정해주었다. 처음에는 조정값의 범위를 거의 주지 않아 정지했을 때 차량이 떨리며 소리가 나는 현상이 있었지만 값을 조절할 때 소리가 나지 않고 중앙정렬을 할 수 있도록 범위를 주어 소음이 나지 않고 중앙으로 회전축을 유지할 수 있도록 알고리즘을 구현하였다.

코드에서 nxt_motor_get_count() > 5, nxt_motor_get_count() <= -1 조정값에 범위를 주었다.

- Slow start

```
//천천히 출발하기 위한 Task, alarm으로 250ms마다
TASK(slowStartTask) {
  if (startstatus == 1 && speed < maxspeed) {
    speed += 5;
  }
  TerminateTask();
}
```

[그림 4] slowstart 구현 코드

```
ALARM slowStartAlarm_250ms {
  COUNTER = SysTimerCnt;
  ACTION = ACTIVATETASK {
    TASK = slowStartTask;
  };
  AUTOSTART = TRUE {
    ALARMTIME = 1;
    CYCLETIME = 250;
    APPMODE = appmode1;
  };
};
```

[그림 5] slowstart Alarm 구현 코드

보통의 차량은 가속 페달을 밟아도 한번에 속도가 증가하지 않았다. 따라서 최대한 실제의 차량과 비슷하게 구현하기 위해서 가속을 해도 천천히 속도가 오르도록 구현하였다. Alarm을 통해 Task를 250ms 마다 실행시켜 전진하며 속도가 바뀔 때만 속도를 점차 올려준다.

- Replay mode

```
if (!record_index) {
  start_play_back = 0;
  record_index = 1;
  display_clear(0);
} else {
  //기록된 속도로 전,후진을 세팅하는 구간
  speed = -record[record_index][0];
  ecrobot_set_motor_speed(NXT_PORT_B, speed);
  ecrobot_set_motor_speed(NXT_PORT_C, speed);

  //기록된 만큼 조향하는 구간
  if (curve_state == 3) {
    nxt_motor_set_speed(NXT_PORT_A, record[record_index][2], 1);
    display_goto_xy(0, 6);
    display_int(1, 1);
    display_update();
  }
```

[그림 6] 차량 동작 record 구현 코드

```
if (record_mode) {
  record[record_index][1] = nxt_motor_get_count(NXT_PORT_B);
  record[record_index][2] = -steering_speed;
  record[record_index++][0] = speed;
}
```

[그림 7] 차량 동작 record 구현 코드

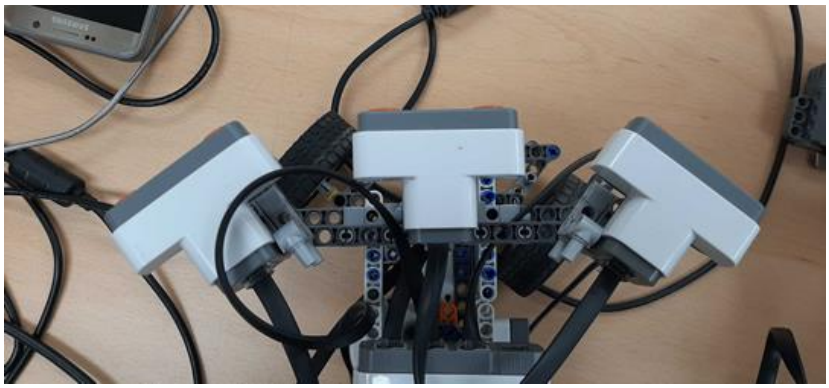
차량의 움직임을 기억했다가 다시 움직임을 거꾸로 플레이백하는 기능이다. 시간을 기록할 수 있는 수단이 없어서 모터가 돌아간 카운트와 속도, 조향 방향등을 기록하여 기록한 배열의 뒤에서부터 처음까지 속도와 조향 방향을 반대로 하여 현재 배열과 이전 배열의 카운트 차이만큼 움직이도록 구현하였다. 이 Task는 Alarm으로 5ms마다 작동하지만 state value를 통해 값이 1일때만 동작하도록 구현하였다.

3.2 Follow Car

3.2.1 H/W

[그림 8] Follow Car

- 조향을 하기 위한 조향 모터 1 개와 주행을 하기 위한 주행 모터 2 개, 추가 기능 및 모터를 움직이기 위한 본체로 이루어진 차체



[그림 9] Follow Car 초음파센서 장착부

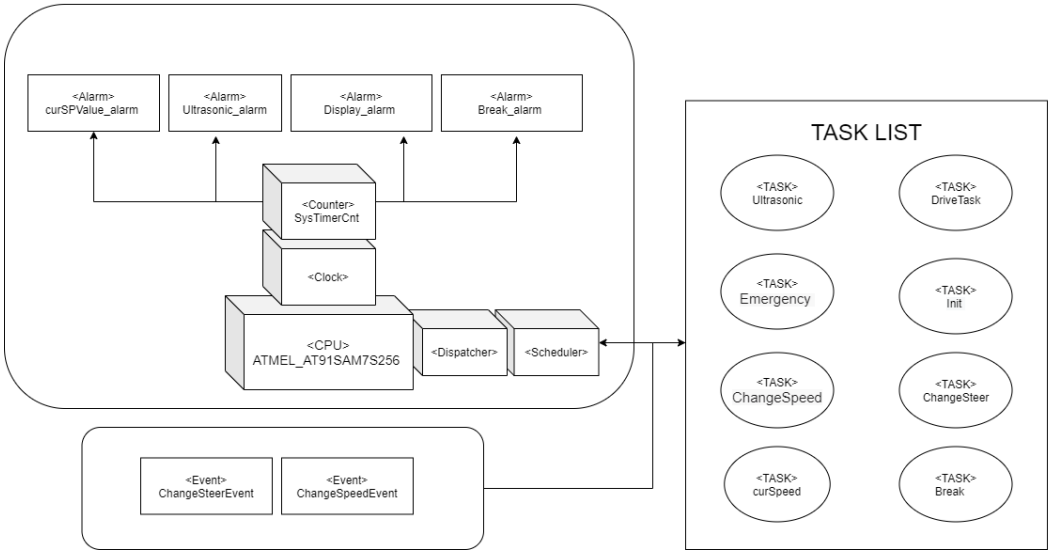
- Control Car 의 운행을 따라가기 위해 Control Car 의 판을 인식할 수 있는 UltraSonic SenSor 3 개로 이루어진 전면부
- UltraSonic SenSor 를 3 개를 쓴 이유는 sensor 간의 간섭을 최소화하고 인식범위를 최대화하기 위함

[그림 10] Follow Car 후면 터치센서 장착부

- 후면 충돌을 인식하기 위한 1 개의 Touch Sensor 로 이루어진 후면 범퍼

3.2.2 S/W

3.2.2.1 OIL Definition



[그림 11] OIL Definition Diagram

3.2.2.2 Task Definition

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	Emergency
PRIORITY	9
TASK TYPE	Basic TASK
IS AUTO START	FASLE
TRIGGER	Break_alarm
FUNCTIONALITY	후면 충돌시 발생하는 Task로 Touch Sensor가 눌린 것을 감지하면 Activated 된다. 발생시 소리를 울리고 전체 OS를 shut down 시킨다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	DriveTask
PRIORITY	5
TASK TYPE	BASIC TASK
IS AUTO START	TRUE
TRIGGER	Display_alarm
FUNCTIONALITY	주행정보를 LCD display에 출력하는 Task로 0.05초 마다

	Display alarm을 통해 실행된다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	UltraSonic
PRIORITY	2
TASK TYPE	BASIC TASK
IS AUTO START	FALSE
TRIGGER	Ultrasonic_alarm
FUNCTIONALITY	Ultrasonic_alarm으로 0.05초마다 UltraSonic Sensor값을 읽고 Filtering해 앞차와의 dist를 구하고 현재 상황을 판별해주는 Task이다. 이를 바탕으로 ChangeSpeedEvent, ChangeSpeerEvent 를 발생시킨다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	ChangeSpeed
PRIORITY	6
TASK TYPE	EXTENDED TASK
IS AUTO START	TRUE
TRIGGER	ChangeSteerEvent

FUNCTIONALITY	UltraSonic Task의 ChangeSpeedEvent로 실행되는 Task로 주행 모터의 속도를 바꾸는 Task이다. Filterling된 UltraSonic Sensor값과 dist를 기반으로 구해진 safe_distance를 이용해 주행 모터의 속도를 변화시킨다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	Init
PRIORITY	9
TASK TYPE	BASICTASK
IS AUTO START	TRUE
TRIGGER	
FUNCTIONALITY	UltraSoincSensor와 조향모터의 Count를 초기화 하는 Task로 초기에 시작돼야 하므로 가장 높은 priority를 가진다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	ChangeSteer
PRIORITY	5
TASK TYPE	EXTENDED TASK

IS AUTO START	TRUE
TRIGGER	ChangeSteerEvent
FUNCTIONALITY	UltraSonic Task의 ChangeSteerEvent로 실행되는 Task로 조향 모터를 조절하는 Task이다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	Break
PRIORITY	7
TASK TYPE	BASIC TASK
IS AUTO START	FALSE
TRIGGER	Break_alarm
FUNCTIONALITY	급정지를 하기 위한 Task로 safestatus가 5가 되면 주행 모터를 멈춘다. 실제 차량의 경우 탑승자의 안전과 직결되므로 Init Task를 제외하고 가장 높은 priority를 가진다.

TASK ATTRIBUTE	ATTRIBUTE VALUE
TASK NAME	curSpeed
PRIORITY	4
TASK TYPE	BASIC TASK

IS AUTO START	FALSE
TRIGGER	curSPValue_alarm
FUNCTIONALITY	현재 속도를 계산하기 위한 Task로 curSPvalue_alarm을 통해 0.05초 마다 실행된다.

3.2.2.3 Function Definition

- void sort()

초음파 센서 Array를 insertion sort하는 함수이다.

```
//초음파 센서 배열에 오름차순으로 정렬
void sort1()
{
  for (int i = 0; i < ArrayLen - 1; i++)
  {
    for (int j = i + 1; j < ArrayLen; j++)
    {
      if (us1array[i] > us1array[j])
      {
        sorttemp = us1array[j];
        us1array[j] = us1array[i];
        us1array[i] = sorttemp;
      }
    }
  }
}
```

[그림 12] insertion sort 구현 코드

- void shutdownFunc()

Follow car의 기능을 정지하기 위한 함수이다.

```
void shutdownFunc(void)
{
    StatusType ex = (StatusType)1;
    ShutdownOS(ex);
}
```

[그림 13] Follow Car 강제종료 구현 코드

3.2.2.4 Intro to Core Algorithm

- UltraSonic Sensor Filtering Algorithm

```
us1array[us1status] = temp1;
us2array[us2status] = temp2;
us3array[us3status] = temp3;
us1status++;
us2status++;
us3status++;

if (us1status == ArrayLen - 1)
{
  //초음파 센서 배열 정렬
  sort1();
  sort2();
  sort3();

  //median 값으로 센서값 정제
  us1 = us1array[1];
  us2 = us2array[1];
  us3 = us3array[1];

  us1status = 0;
  us2status = 0;
  us3status = 0;
}
```

[그림 14] 초음파 센서 값 정제 알고리즘

위 코드에서 정밀한 값을 추출하기 위해 UltraSonic Sensor의 값을 Array size가 4인 usarray에 저장을 한다.

그리고 insertion sort 를 사용해 Array를 정렬하고 Median 값을 사용한다. 그리고 시연 환경에서 책상, 벽과 같은 장애물을 인식할 수 있기 때문에 UltraSonic Sensor값이 100 이상일 경우 Control Car를 인식하지 못했다고 처리한다.

- Speed Control Algorithm according to Safe Distance Rule

앞차와 부딪히지 않고 일정하게 거리를 유지하기 위해서는 적절한 시점에 속도를 줄일 수 있어야 한다. 적절한 시점은 실제 차량에서 '제동거리'로 존재 한다. 제동거리는 다음 수식을 따라서 크기가 변한다.

$$d \propto \ motor\ speed * \frac{1}{kinetic\ friction} * weight$$

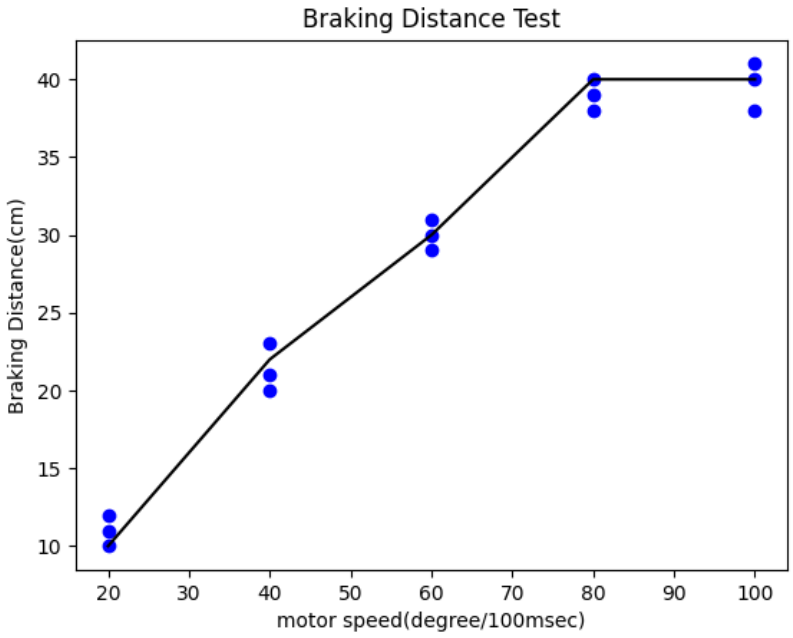
[그림 15] 제동거리 계산 공식

수식에 따르면 현재 속도에 비례하고 바닥과의 마찰력에 반비례하는 것을 알 수 있다. 이외에도 영향을 주는 조건은 브레이크 성능, 바닥의 상태에 따라서 달라진다. 이점을 기인하여 실제로 시연이 이루어질 장소에서 각각의 속도에서 정지를 했을 때 발생하는 제동거리가 얼마나 되는지 실험했다.

실험 조건 :

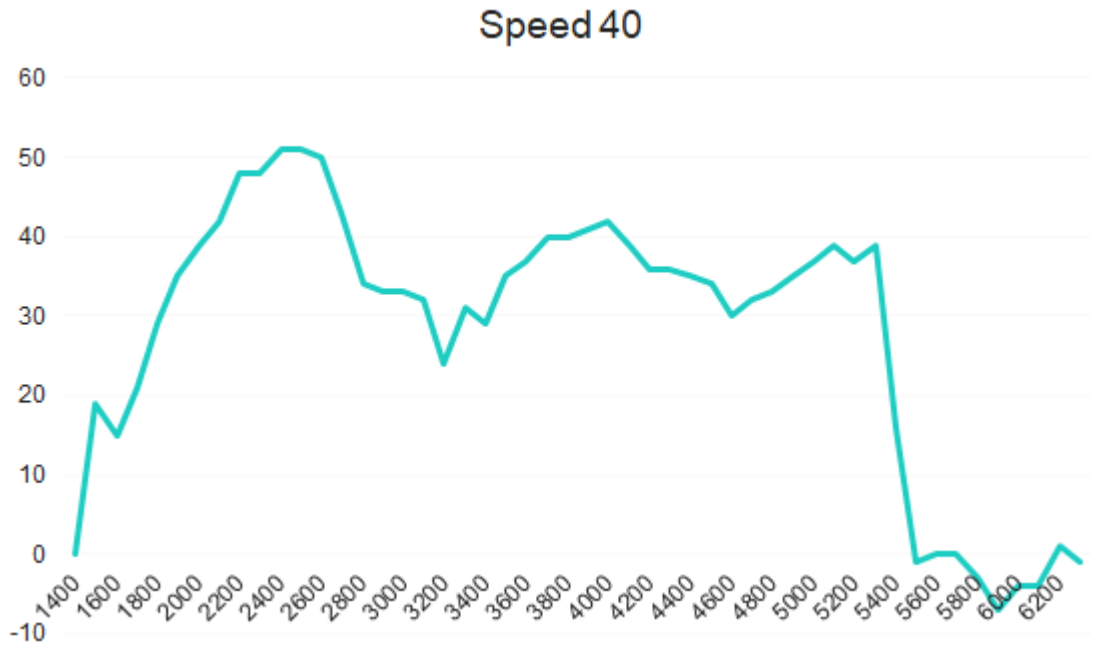
1. 실제 시연할 공간에서 실험 한다.
2. 뒷차로 쓸 차량으로 설정 속도에 도달 할 때까지 최대한 직선으로 달린다.
3. 자동차가 설정 속도에 도달할 때 급정지 한다.
4. 급정지한 시점부터 정지할 때까지의 제동거리를 구한다.

실험 결과는 다음과 같다.

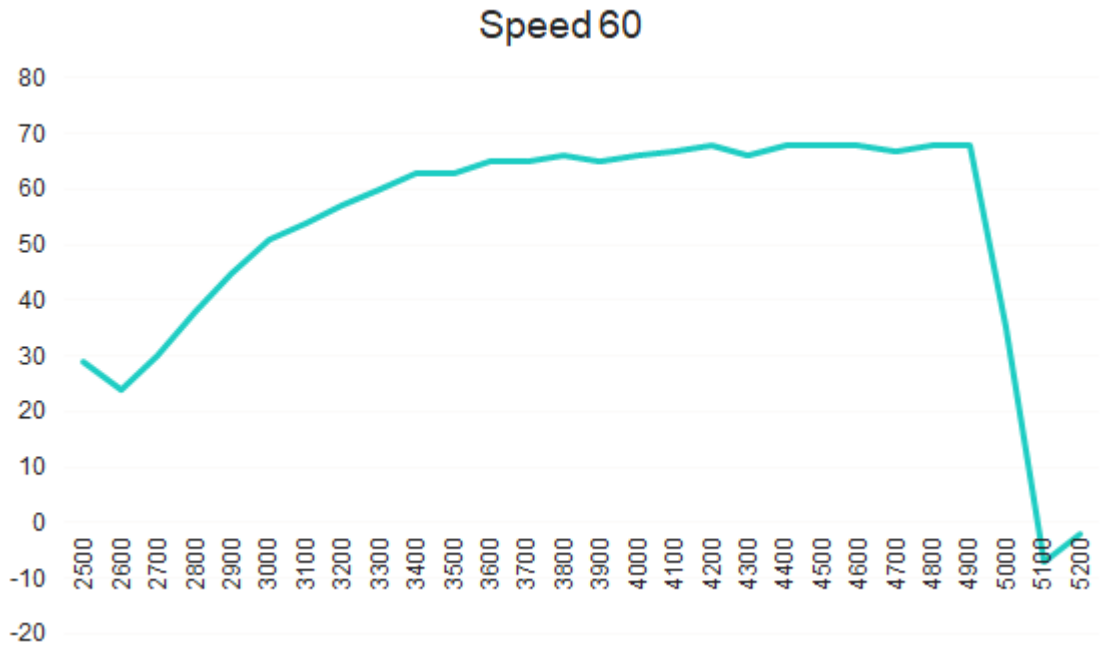


[그림 16] 실제 제동거리 실측 값

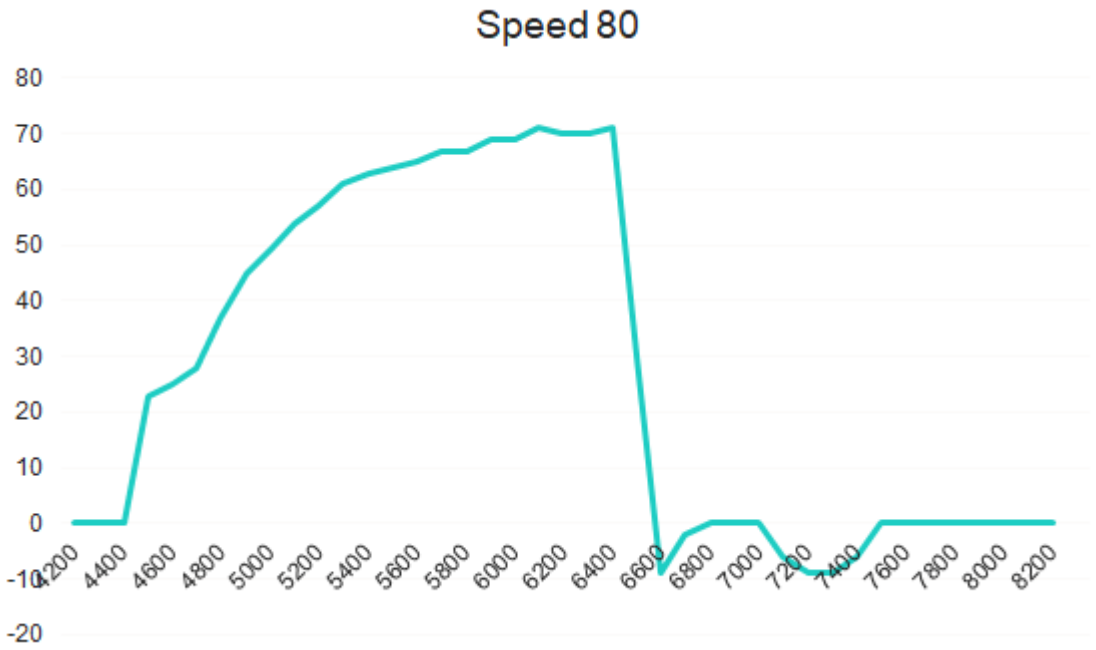
다음 그래프를보면 제동거리는 위에서 말한 공식을 따른 다는 것을 알 수 있었다. 하지만 한가지 이상한 점이 발견되었다. 제동거리가 속도가 80이상일 경우 더이상 증가지 않는다는 것이다. 이를 확인하기 위해서 NXT Log Files이라는 기능을 이용해서 실제 바퀴가 돌아가는 속도를 측정해보았다.



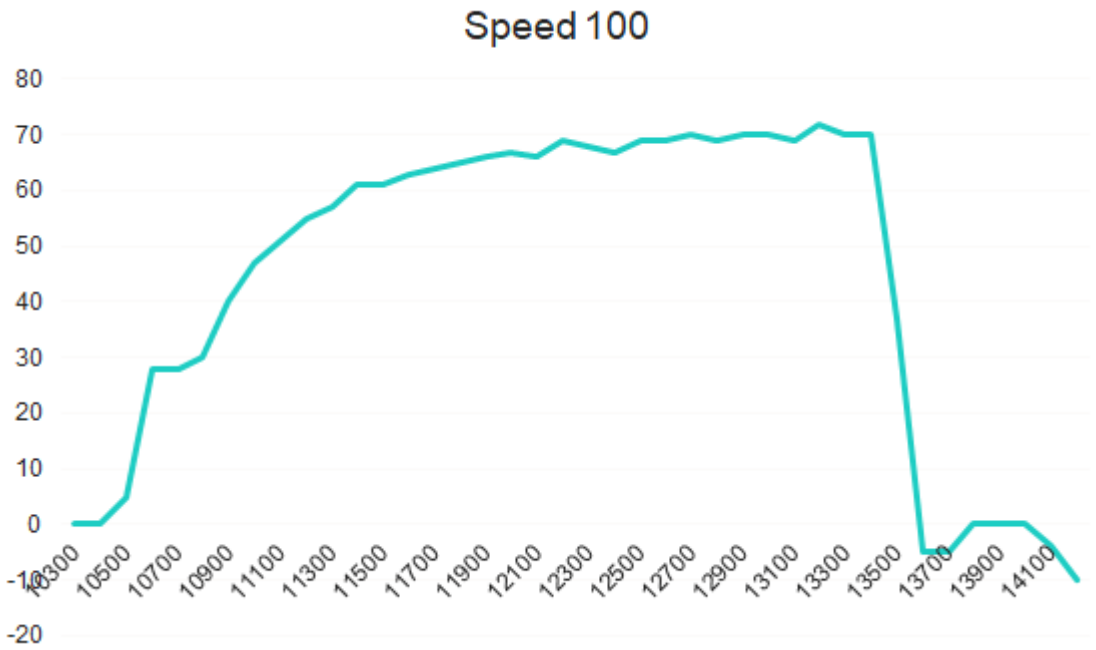
[그림 17] Motor Speed 40의 측정 시간에 따른 실제 Motor Speed 그래프



[그림 18] Motor Speed 60의 측정 시간에 따른 실제 Motor Speed 그래프



[그림 19] Motor Speed 80의 측정 시간에 따른 실제 Motor Speed 그래프



[그림 20] Motor Speed 100의 측정 시간에 따른 실제 Motor Speed 그래프

위 그래프는 각 속도마다 실제로 모터가 돌아가는 속도를 나타낸 것이다. 실제로 75 이후로는 모터가 회전하는 속도가 더이상 증가하지 않는데 이는 차량의 뒷바퀴와 연결되 모터가 현재 차량의 무게를 가지고 움직일 때 움직일 수 있는 속도의 제약 조건이 있기 때문이다. 위 그래프를 통해서 다음과 같은 사실을 알아 냈다.

첫번째, 현재 모터가 돌아가는 속도와 설정한 스피드에 도달하기 위해서는 다음과 같은 차이가 걸린다. 이 차이는 실제 RTOS에서 치명적이므로 제동거리는 반드시 실제 스피드를 기준으로 정해져야 한다. 모터가 실제로 돌아가는 속도를 구하는 방법은 NXT Log Files을 보고 참고 했고 다음과 같은 공식을 따른 다는것을 알 수 있었다.

Sync data	0	0	1586929	100	-1
Sdata	6_FP Rotation Sensor_deg				
Time	FP Rotation Sensor				
0	0				
100	0				
200	0				
300	0				
400	0				
500	0				
600	0				
700	0				
800	0				
900	0				
1000	0				
1100	0				
1200	0				

[그림 21] 시간에 따른 실측 속도 Log 기록 값

Sync data	0	0	1312355	100	-1
Sdata	6_FP Rotation Sensor_deg			
Time	FP Rotation Sensor			
10600	-5			
10700	-33			
10800	-61			
10900	-91			
11000	-131			
11100	-178			
11200	-229			
11300	-284			
11400	-341			
11500	-402			
11600	-463			
11700	-526			
11800	-590			

[그림 22] 시간에 따른 실측 속도 Log 기록 값

$$power\ unit = (\frac{degree}{100msec})$$

[그림 23] Motor Power 측정 단위

하지만 100msec마다 측정하는것은 옳지 않았다. 실제로 100msec단위로 차량의 속도를 측정할 경우 실제 속도와 오차가 있었다. 따라서 이 값을 50msec로 줄이고 다음과 같은 코드로 구현 했다.

```
TASK(curSpeed) {
  curSPValue = (before_count - nxt_motor_get_count(B)) * 2;
  before_count = nxt_motor_get_count(B);
  TerminateTask();
}
```

[그림 24] 주행 차량의 실제 속도 계산 알고리즘

둘째, 현재 모터의 힘과 차량의 무게를 고려했을 때 차량의 최대 속도는 80로 제한되어 있다. 따라서 차량의 제동거리는 차량의 속도가 20~80일때를 기준으로 정한다. 속도-제동거리 그래프를 통해서 속도와 제동거리와의 관계를 공식으로 도출했다.

$$braking\ distance = 20 + \frac{(current\ Motor\ Speed - 20)}{60} * 40$$

[그림 25] 속도에 따라 변화하는 제동거리 계산 알고리즘

기울기가 20이 아닌 40인 이유는 앞차가 갑자기 방향을 바꿀 때 대각선 방향의 소나 센서가 실제 거리보다 크게 측정해서 앞차와의 거리가 가까움에도 부딪칠 위험이 있기 때문이다. 이 점을 고려해서 다음과 같은 방식으로 코드를 구현했다.

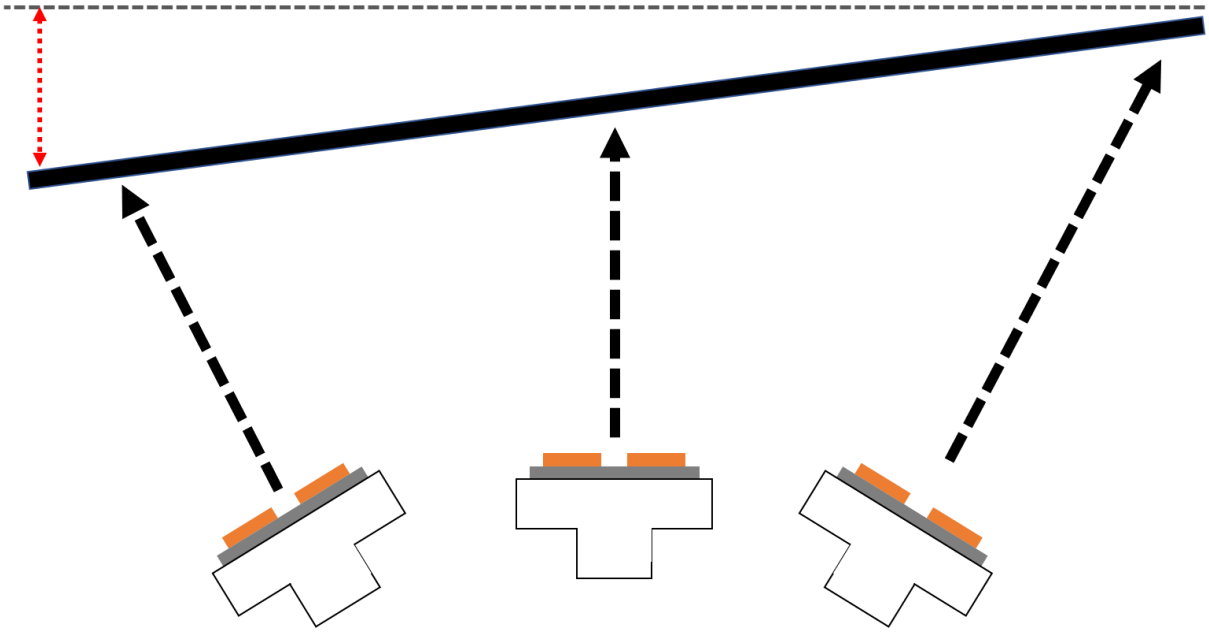
```
if (curSPValue < 20) {
  safe_distance = SAFE_DISTANCE_BASE;
} else {
  safe_distance = SAFE_DISTANCE_BASE +
                  SAFE_DISTANCE_RANGE * fabs((curSPValue - 20) / 60.0);
}
```

```
if (dist < safe_distance) {
  spvalue = 0;
  if (dist < 10) {
    spvalue = -30;
  }
} else if (dist < safe_distance + 10) {
  spvalue = 35;
} else {
  spvalue = 65;
}
```

[그림 26] 속도에 따라 변화하는 안전거리 계산 구현 코드

- Steering Control Algorithm Using Ultra sonic sensor

초음파센서 세개를 사용하여 더욱 정밀하게 조향기능을 구현하였다. 구간을 크게 미세조정 범위, 커브 범위, 급커브 범위 세가지로 나누었다. 구현 방식은 다음과 같다.



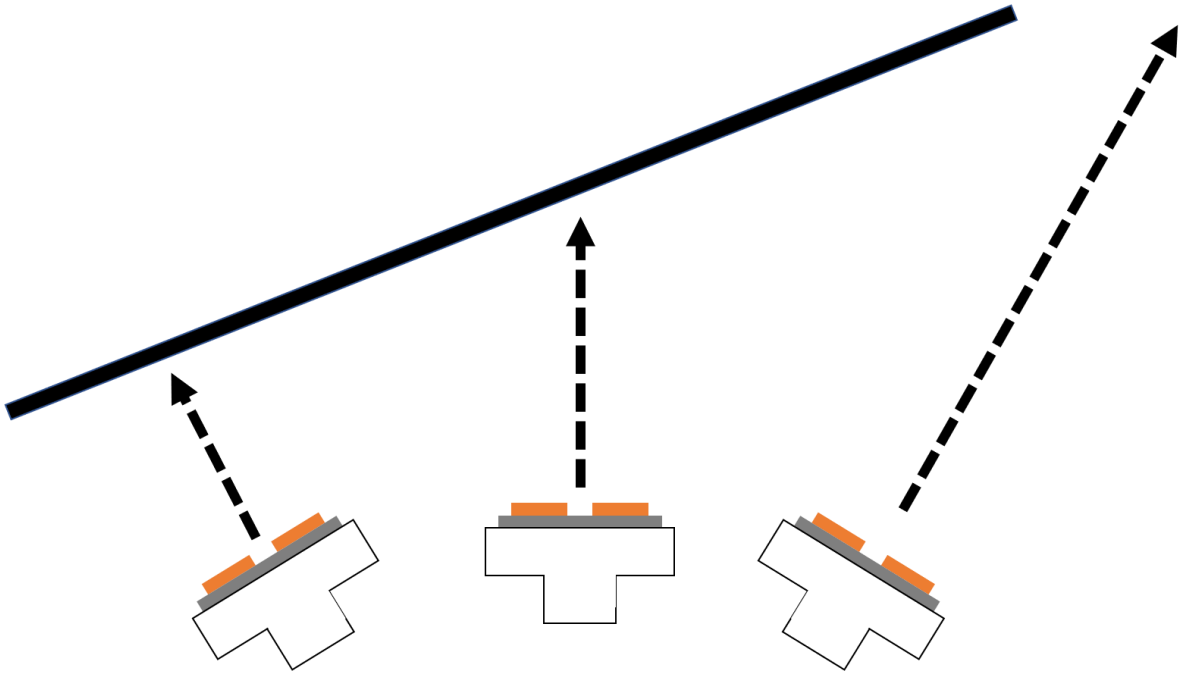
[그림 27] 미세 조향 범위

미세 조정 범위는 좌 우측 및 전방 세개의 센서가 모두 전방의 물체와의 거리를 측정 가능한 경우를 말한다. 이때의 조향값은 좌측과 우측의 측정값 차이에 비례하여 증가한다. 본 프로젝트에 이용된 차량에서는 이탈값 없이 최대 8cm 차이까지 측정 가능하여 2cm 차이부터 8cm 차이까지 총 7단계로 미세 조향을 구현하였다.

```
else //모든 수치 있는 우회전 -> 미세 조향
{
  if ((us1 - us3) * 3 >= 24)
  {
    motorcount = maxmotorcount - 6;
  }
  else
  {
    motorcount = (us1 - us3) * 3;
  }
}
```

[그림 28] 미세 조향 구현 코드

위 코드에서 us1은 좌측 측정값을 , us3은 우측 측정값을 의미한다. 두 측정값의 차에 상수 3을 곱해주는 것을 알 수 있는데, 이는 여러번의 실측을 통해 얻어낸 결과로 전방 차량과 가장 비슷하게 조향하는 수치이다.



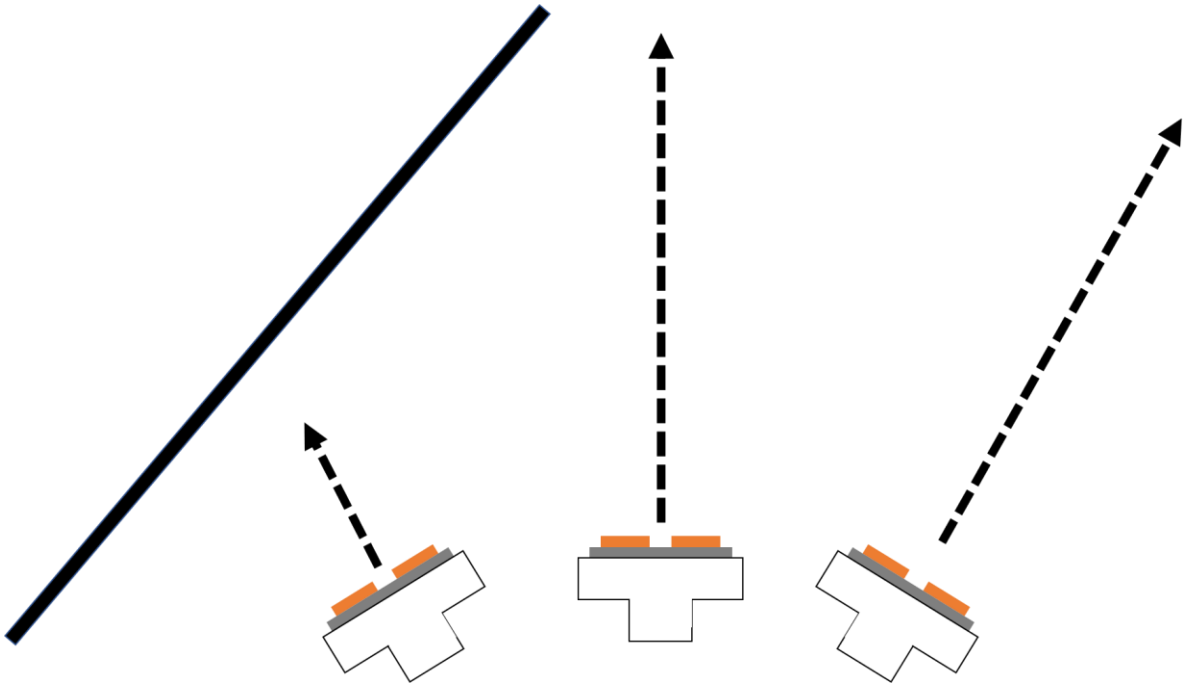
[그림 29] 커브 조정 범위

커브 조정 범위는 좌측 또는 우측의 센서가 값을 측정할 수 없는 경우를 말한다. 이는 미세 범위의 최대 측정차인 8cm 를 벗어난 경우로 완전한 좌회전 또는 우회전을 의미한다. 전방 차량과의 거리가 측면 초음파 센서의 측정 범위를 벗어나게 되면 위 그림과 같이 측정 불가 센서의 반대 방향으로 조향했다는 것을 알 수 있다.

```
if (us1 > 100) //us1 수치 잃음 -> 우회전
{
  if (safestatus == 4) //us1, us2 수치 잃음 -> 오른쪽 급커브
  {
    motorcount = maxmotorcount;
  }
  else //us1 수치 잃음 -> 오른쪽 커브
  {
    motorcount = maxmotorcount - 5;
  }
}
```

[그림 30] 커브 범위 조향 구현 코드

위 코드에서 한쪽의 수치를 잃었을 경우 최대 모터 회전 수인 maxmotorcount보다 5만큼 적게 회전하는 것을 알 수 있다.



[그림 31] 급커브 조향 범위

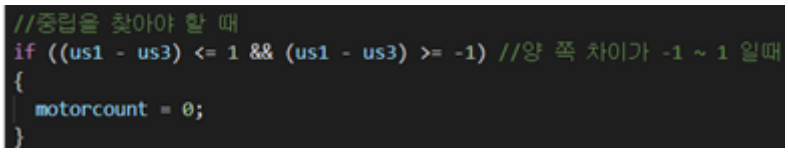
급커브 조정 범위는 좌측과 중앙 또는 우측과 중앙 총 두개의 센서가 값을 측정할 수 없는 경우를 말한다. 이는 커브 범위를 벗어난 급커브를 의미한다. 위 그림에서 전방 차량이 한쪽 측면 센서에서만 감지 되므로 급커브를 한 것을 알 수 있다.

```
if (us1 > 100) //us1 수치 잃음 -> 우회전
{
  if (safestatus == 4) //us1, us2 수치 잃음 -> 오른쪽 급커브
  {
    motorcount = maxmotorcount;
  }
```

[그림 32] 급커브 조향 구현 코드

위 코드에서 safestatus 4 는 중앙과 한쪽 측면 센서 총 두개의 센서가 측정 불가 상태임을 의미하고, 전방 차량이 급커브 조향을 한 것으로 판단하여 최대 모터 회전수인 maxmotorcount 까지 조향을 하는 것을 알 수 있다.

Follow 차량은 좌우 조향이 종료되면 직진을 위해 정면으로 조향하게 되는데 다음과 같이 구현하였다.



```
//중립을 찾아야 할 때
if ((us1 - us3) <= 1 && (us1 - us3) >= -1) //양 쪽 차이가 -1 ~ 1 일때
{
  motorcount = 0;
}
```

[그림 33] 직진 인지 코드

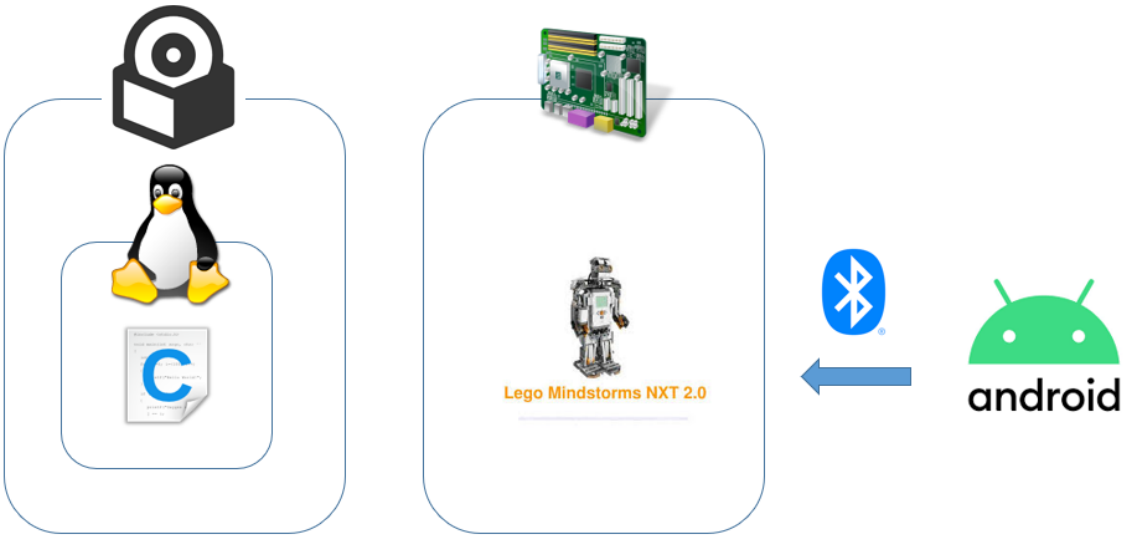
좌우 측정값이 -1cm ~ 1cm 사이일때는 전방 차량이 직진 주행을 하고 있다고 판단하고 모터의 목표 회전 값을 0으로 변경한다.

```
else if (motorcount == 0) //중립 -> 중앙(momtorcount = 0)을 찾아감
{
  if (nxt_motor_get_count(A) > motorcount)
  {
    ecrobot_set_motor_speed(A, -steerspeed);
  }
  else if (nxt_motor_get_count(A) < motorcount)
  {
    ecrobot_set_motor_speed(A, steerspeed);
  }
  else
  {
    ecrobot_set_motor_speed(A, 0);
  }
}
```

[그림 34] 중앙 정렬 구현 코드

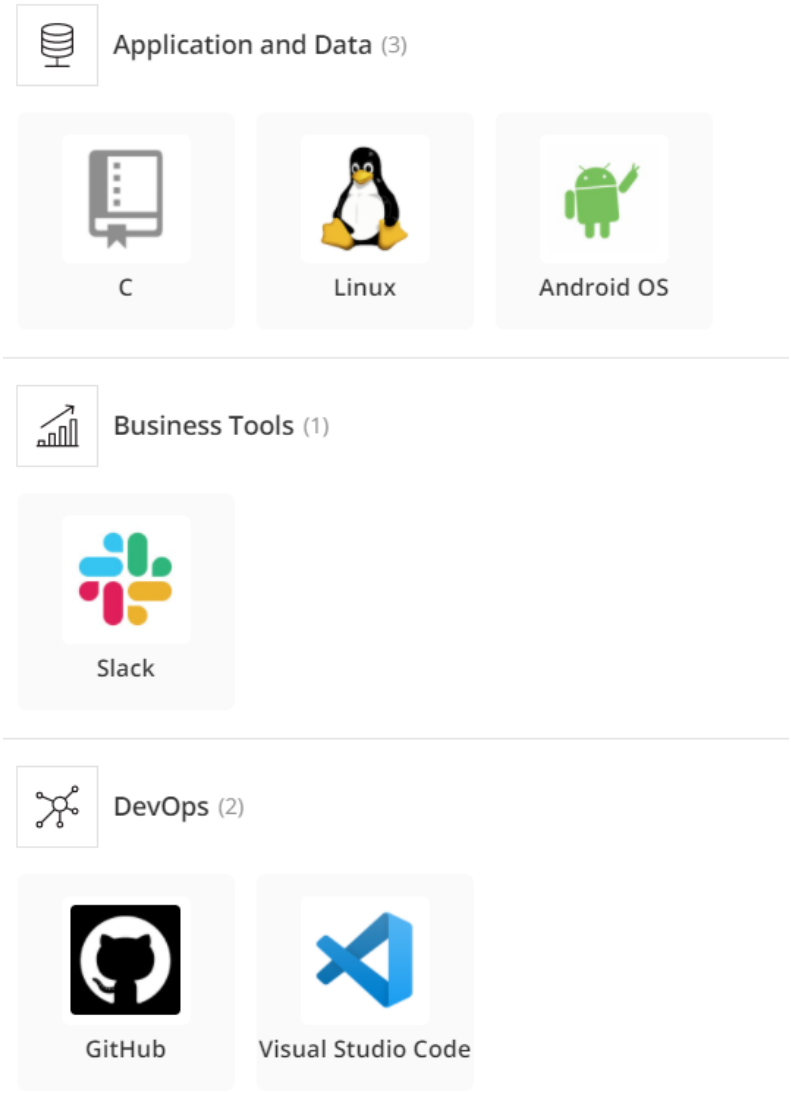
모터의 회전값이 0일 경우 실시간으로 실제 모터 회전값을 nxt_motor_get_count() 함수를 이용하여 받아온다. 실제 모터 회전값이 0보다 클 경우 0이하로 회전시키고, 0보다 작을 경우 0 이상으로 회전시킨다. 실제 회전값이 0에 수렴할 때까지 위 동작을 반복하여 최대한 실제 모터 회전값을 0에 가깝게 유지시킴으로써 중앙 정렬 기능을 구현하였다.

3.2.3 시스템 구성



[그림 35] 시스템 구성도

3.2.4 사용 소프트웨어



[그림 36] 사용 소프트웨어 스택

4 결과 및 분석

4.1 프로젝트 결과

- Control Car

요구사항	내용	구현 결과
직진 / 후진 기능	모터를 제어하여 차량의 직진 및 후진을 구현한다.	구현 완료
좌 / 우 조향 기능	모터를 제어하여 차량의 좌회전 및 우회전을 구현한다.	구현 완료
브레이크 기능	모터를 제어하여 급브레이크 기능을 구현한다.	구현 완료
고속 / 저속 기능	모터를 제어하여 고속 및 저속 주행 기능을 구현한다.	구현 완료

- Follow Car

요구사항	내용	구현 결과
앞 차 출발시 같이 출발 기능	앞차가 출발 했을 때 따라서 달릴 수 있는 기능 구현	구현 완료
앞 차와 거리 유지 기능	앞차가 속도를 변경 했을 때 앞차와의 거리를 조정하는 기능 구현	구현 완료
급제동 기능	앞차가 급제동/천천히 멈출 때 맞춰서 속도를 줄이는	구현 완료

	기능 구현	
조향 기능	앞차의 방향에 맞춰서 방향을 바꾸는 기능 구현	구현 완료
급 회전 기능	앞 차를 잃었을 때 따라가는 기능 구현	구현 완료
후진 기능	앞 차가 후진 할때 부딪히지 않고 같이 후진 하는 기능 구현	구현 완료

- 추가 기능

구현사항	내용	구현 결과
전방 차량이 사라졌을 때 대처 기능	전방 차량이 사라졌을 경우 이전 주행 상태를 유지하여 사라진 차량을 찾는다.	구현 완료
후방 충돌 인지 기능	차량 후방에서의 충돌을 인지한 후 차량을 긴급제동한다.	구현 완료

4.2 개선할 점

4.2.1 H/W

- 다른 팀들과 다르게 큰 타이어를 사용했는데 작은 타이어에 비해 큰 저항력을 가져 큰 힘이 필요하기 때문에 섬세한 속도조절이 힘들다는 단점이 있었다. 작은 타이어로 바꾼다면 더욱 조향과 주행이 섬세해질 것이며 부드러운 움직임을 보일 것이다.

- Ultra sonic Sensor 2개와 가운데에 Light Sensor를 이용하려 했지만 UltraSonic Sensor 자체도 정밀하지 않았고 Light Sensor도 환경적인 한계가 있어 사용하지 못했다. 정밀한 UltraSonic Sensor로 교체를 한다면 더욱 정밀한 군집주행이 가능할 것이다.

4.2.2 S/W

- REAL TIME CONSTRAINT and SAFETY and EFFICIENT

Realtime Constraint	차량에 들어가는 임베디드 시스템이기 때문에 ms단위의 짧은 반응 속도를 만족해야 한다. nxt lego mindstorm의 경우에 Sensor issue로 인해 특정 시간아래로는 반응속도를 줄일 수 없기에 더 빠른 반응속도를 위해서 기본으로 제공되는 Sensor 말고 다른 Sensor를 이용해야 할 것이다.
energy efficient	
memory efficient	현재 OSEK OS를 통해서 차량을 조작하는데 어플리케이션에서 사용하는 최대 메모리가 1KB가 안된다. 이 수치는 OSEK OS에 크게 영향을 미치지 않는 값이라고 판단된다.
Safety Problem	차량은 승객의 안전이 최우선 이기 때문에 항상 충돌을 방지해야 한다. 이를 위해서 급 제동, 천천히 제동, 후방 충돌 감지를 소프트웨어, 하드웨어 적으로 구현 하였다. 하지만 이것만 가지고 완벽하게 Safety Problem을 해결했다고 할 수 없을 것이다. 거리를 감지하는데 다른 센서를 사용하거나 혹은 제동거리를 좀더 세밀하게 측정하면 더 좋은 결과를 얻을 수 있을 것이다.

- EMBEDDED SYSTEM CHARACTER REVIEW

reliability	뒷차의 차량이 앞차의 차량과 충돌을 방지하는 기능이 잘 구현되어 있다. 하지만 급커브시에 멈춤에도 불구하고 앞차와 살짝 맞닫는 상황이 발생한다. 이는 소프트웨어적으로 해결할 수 있는 부분이다.
maintainability	차량의 설계상 3개의 모터 4개의 센서를 사용한다. 이는 큰 배터리 소비이다.
availability	차량의 배터리가 없을 경우 제대로 작동하지 않을 경우가 있다.
security	이상 없음.

4.3 소회

집중교육 이전의 프로젝트들은 소프트웨어만 고려하여도 충분히 좋은 결과를 만들어낼 수 있었지만 Osek과 같은 Real-time OS 임베디드 프로젝트들은 소프트웨어 뿐 아니라 하드웨어의 특성 또한 고려해야하기 때문에 기존의 생각방식과는 다르게 다각화하여 복잡한 사고를 필요로 함을 느꼈다. 예를 들면 nxt lego mindstorm은 건전지 전압에 따라 motor power가 큰 폭으로 달라지기 때문에 남은 배터리에 따라 같은 입력값을 주어도 매번 다른 속도로 모터가 동작하여 새로운 결과값을 도출했다. 또한, 센서로 얻어지는 값도 정밀하지 않아 소프트웨어적으로만 계산하는 것은 부정확했다. 따라서 측정값을 그대로 사용하기 보다는 측정값을 이용해 실제로 사용할 수 있는 방향을 생각해보아야 했고, 주어진 조건을 이용하여 실측하여 사용하는 방법을 택했다. 그 결과 기존의 방법보다 더 정확한 값을 얻을 수 있었다. 조향과 속도 조절면에서 이러한 방법을 택하여 값을 구하였다.

조향에서는 UltraSonic Sensor의 정밀도가 문제였다. 값을 일정하게 받지 못하고 일직선으로 배치하면 서로에게 간섭을 주어 조향모터를 섬세하게 조절하기 어렵다. 이 점을 고려해 3개의 Sensor를 사용했으며 Sensor의 배치를 전방만 바라보는 것이 아닌 전방과 좌,우 대각선으로 조립하여 서로의 간섭을 최대한으로 줄였고 앞 차가 회전을 하더라도 잘 따라잡을 수 있도록 하였다. 그 결과 전방만 바라보는 두 개의 센서를 이용했을 때보다 더욱 좋은 결과를 얻을 수 있었다.

속도 조절에서는 배터리와 차량의 무게에 따라서 일정 값의 속도를 모터에 전해주어도 실제로는 그 속도가 나오지 않는다는 것이 문제점으로 다가왔다. 실제 속도를 알지 못하면 제동거리를 알 수 없어서 앞 차와 부딪히거나 제대로 따라가지 못하는 상황이 벌어진다. 따라서 속도 조절은 매우 중요한 요소인데 여러번의 실험을 통해 움직이는 차량의 실제 속도를 구해낼 수 있었다. 일정 값으로 차량을 돌려 로그를 찍어본 결과 100ms마다 모터회전 수의 변화량을 보면 차량의 실제 속도와 비슷해지는 것을 알 수 있다. 따라서 Alarm과 모터의 회전수를 통해 차량의 실제 속도를 구하고 속도에 따른 제동거리를 구할 수 있었다. 결과적으로 차량은 일정한 거리를 유지하며 따라가고 속도가 빨라도 앞 차와 부딪히지 않고 멈출 수 있었다.