

2018년 1학기 로봇프로그래밍 라인트레이서 포트폴리오



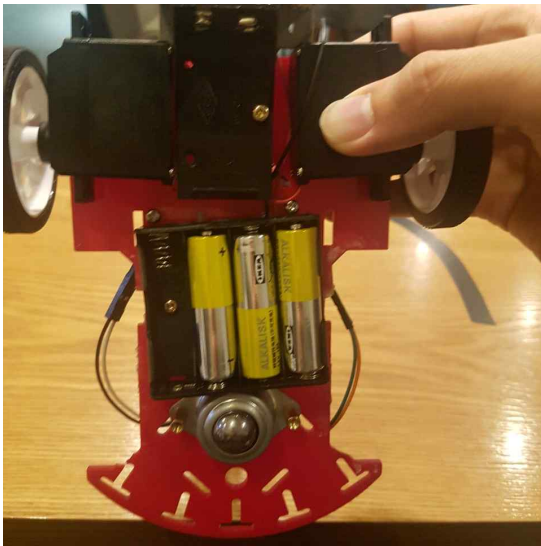
2016110576
기계로봇에너지공학부
강재원

목차

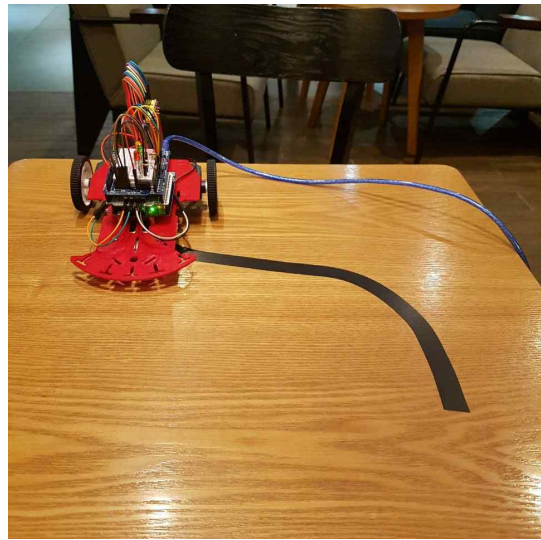
I. 기초 골격의 조립.....	2p
II. 외관의 디자인.....	3p
III. 코딩 및 디버깅.....	4p
1. 코드 전문 및 설명	
2. 메인 코드의 변화	
3. 보정 코드의 변화	
IV. 한계점 및 개선방안.....	10p

I. 기초 골격의 조립

기본적인 골격의 모습은 일반적인 RC카의 모습을 따랐다. 그러나 모터와 본체의 플라스틱 판을 접착시킬 때, 양면테이프를 사용하였다. 이는 조립의 시간을 단축시켜주었고, 모터의 위치를 바꿀 때에도 용이했다. 또한, 차체의 앞부분을 바닥과 최대한 밀착시켜 센서가 라인을 보다 잘 인식하도록 디자인하였다.

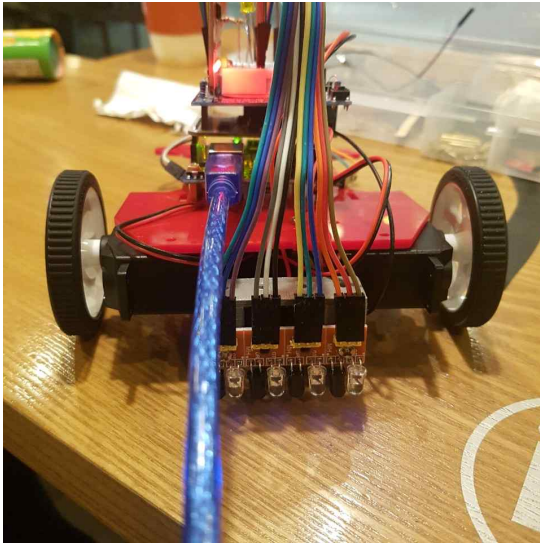


라인트레이서의 밑면 사진

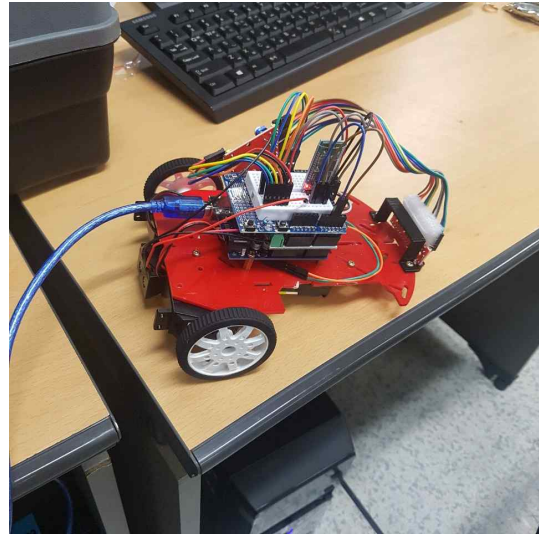


디버깅용 LED를 활용해 코딩을 하는 모습

가장 고민이 되었던 점은 ‘센서의 위치’, 즉 모터와 먼 쪽에 센서를 부착할 것인지, 또는 센서와 가까운 쪽에 센서를 부착할 것인지에 대한 것이었다. 모터와 가까운 쪽에 설치할 경우, 차체의 회전 중심이 라인의 중심과 크게 멀지 않아 더욱 안전한 운행을 할 수 있었다. 반면 모터와 먼 쪽에 설치할 경우, 차체가 회전하는데 필요한 모터의 힘이 적어 더 빠른 속력을 낼 수 있었다. 두 경우 모두 실행해본 결과, 센서를 모터와 먼 쪽에 설치하여도 안정성을 충분히 확보할 수 있었기 때문에 모터와 먼 쪽에 설치하기로 하였다.



센서를 모터와 가까운 쪽에 부착시켜보았을 때의 사진

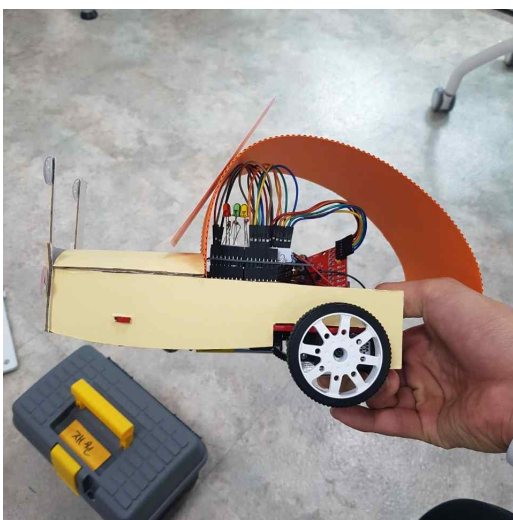


최종적으로 완성된 골격의 모습

II. 외관의 디자인

라인트레이서의 외관 디자인 방향은 홀로 고심한 끝에 '달팽이'를 모티브로 삼기로 하였다. 평소에 즐겨하던 게임에 달팽이를 닮은 캐릭터가 있어, 이를 오마주하기로 하였다.

베이지색의 하드보드지를 사용하여 달팽이의 몸과 머리를 표현하였고, 골판지를 사용하여 달팽이의 등껍질을 표현하였다. 그러나, 라인트레이서에 디버깅한 코드를 지속적으로 업로드 해야 하고, 배터리 방전을 피하기 위해 주행 중이 아닐 때에는 배터리 선을 뽑아놓아야 하므로 컴퓨터 연결선과 필자의 손이 들어갈 공간이 필요했다. 이에 대한 해결책으로 달팽이 등껍질의 뒷공간이 열릴 수 있도록 디자인하였다.



III. 코딩 및 디버깅

1. 코드 전문 및 설명

다음은 필자가 라인트레이서에 사용한 최종코드의 전문이다.

```
//모터변수
int m1=5, m2=6, e1=4, e2=7;
double add_value = 0;
//1R센서 변수
int sensor1=9, sensor2=10, sensor3=11, sensor4=12;
int sensor1_v, sensor2_v, sensor3_v, sensor4_v;
//스위치,상태 변수
int state=0;
int switch_pin=13, switch_value, switch_yes=1;

void setup()
{
    pinMode(m1,OUTPUT);
    pinMode(m2,OUTPUT);
}

void loop()
{
    re:
    sensor1_v=digitalRead(sensor1);
    sensor2_v=digitalRead(sensor2);
    sensor3_v=digitalRead(sensor3);
    sensor4_v=digitalRead(sensor4);

    switch_value=digitalRead(switch_pin);
    if(switch_value==0)//스위치를 누르면
    {
        if(switch_yes==1)
        {
            state = 1-state;
            switch_yes=0;
        }
    }
    if(switch_value==1)//스위치에서 손떼면
    {
        switch_yes=1;
    }

    if(state==0)
    {
        analogWrite(m1,0);
        analogWrite(m2,0);
    }

    if(state==1)
    {
        if(sensor1_v+sensor2_v+sensor3_v+sensor4_v==0)
        {
            if(add_value<0)
            {
                digitalWrite(e1,LOW);
                digitalWrite(e2,LOW);
                analogWrite(m1,225-add_value);
                analogWrite(m2,50);
            }
            if(add_value>0)
            {
                digitalWrite(e1,HIGH);
                digitalWrite(e2,HIGH);
                analogWrite(m1,50);
                analogWrite(m2,225+add_value);
            }
            goto re;
        }

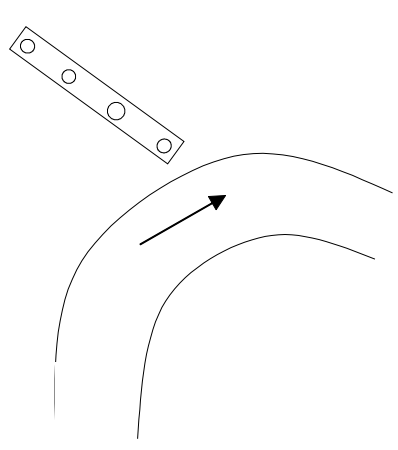
        add_value =
        (sensor1_v*(-60)+sensor2_v*(-30)+sensor3_v*30+sensor4_v*60)/(1+sensor1_v+sensor2_v+sensor3_v+sensor4_v);

        digitalWrite(e1,LOW);
        digitalWrite(e2,HIGH);
        analogWrite(m1,225-add_value);
        analogWrite(m2,225+add_value);

        if(sensor1_v+sensor2_v+sensor3_v+sensor4_v==4)
        {
            state=0;
        }
    }
}
```

이제 코드를 부분별로 자세하게 살펴보자.

코드	설명
<pre>int m1=5, m2=6, e1=4, e2=7; double add_value =0;</pre>	모터를 제어하기 위해서 선언한 변수들이다. add_value에 경우 뒤의 메인코드에서 사용할 변수인데, 전역변수로 선언하기 위해 여기서 선언하였으며, 연산중 나눗셈이 진행되므로 실수 값을 얻기 위해 double 자료형으로 선언하였다.
<pre>int sensor1=9, sensor2=10, sensor3=11, sensor4=12; int sensor1_v, sensor2_v, sensor3_v, sensor4_v;</pre>	IR센서에서 값을 얻기 위해 선언한 변수들이다.
<pre>int state=0; int switch_pin=13, switch_value, switch_yes=1;</pre>	On/Off 스위치를 제어하기 위해 선언한 변수들이다.
<pre>void setup() { pinMode(m1,OUTPUT); pinMode(m2,OUTPUT); }</pre>	setup 함수 안에서는 모터제어를 위해 핀 모드를 설정하였다.
<pre>void loop() {</pre>	이제 loop안을 살펴보자.
<pre>re: sensor1_v=digitalRead(sensor1); sensor2_v=digitalRead(sensor2); sensor3_v=digitalRead(sensor3); sensor4_v=digitalRead(sensor4);</pre>	변수들에 센서가 읽은 값을 저장하는 코드이다.
<pre>switch_value=digitalRead(switch_pin); if(switch_value==0) { if(switch_yes==1) { state = 1-state; switch_yes=0; } } if(switch_value==1) { switch_yes=1; }</pre>	On/Off스위치를 제어하는 코드이다. switch_yes는 스위치를 꺾 눌렀을 때 state가 여러 번 변하지 않고 한번만 변하게 하도록 하는 장치로, 코드의 맨 처음에 1로 초기화하였다. 스위치를 눌렀을 때(switch_value가 0일 때), 누른 순간 두 번째 if문 안으로 들어가 state를 반전시키고 switch_yes의 값을 0으로 바꾼다. 따라서 스위치를 뺐다 다시 누르기 전까지는 두 번째 if문 안으로 들어가지 않는다.
<pre>if(state==0) { analogWrite(m1,0); analogWrite(m2,0); }</pre>	state가 0일 때는 두 개의 모터 모두가 작동을 멈추고, 아두이노 또한 센서 값을 변수에 지속적으로 저장할 뿐 다른 일은 하지 않는다.
<pre>if(state==1) {</pre>	state가 1일 때, 즉 라인트레이서가 작동할 때의 코드를 알아보자.
<pre>add_value= (sensor1_v*(-60)+sensor2_v*(-30)+sensor3_v*30+sensor4_v*60)/(1+sensor1_v+sensor2_v+sensor3_v+sensor4_v);</pre>	위의 코드 전문에서 보면 왼쪽의 코드보다 아래의 if문이 먼저 나온다. 그러나 왼쪽의 코드를 먼저 설명해야하기 때문에 먼저 다뤄보자. add_value의 경우 첫 번째 센서에는 -60, 두 번째 센서에는 -30, 세 번째 센서에는 30, 네 번째 센서에는 60의 값을 할당하고, 그 값들은 더

<pre>digitalWrite(e1,LOW); digitalWrite(e2,HIGH); analogWrite(m1,225-add_value); analogWrite(m2,225+add_value);</pre>	<p>한 값을 $(1+sensor1_v+sensor2_v+sensor3_v+sensor4_v)$로 나눈다. 센서마다 할당해준 값들은 여러번 시험주행을 해보며 가장 안정적인 값을 고른 것이며, 1을 더해준 것은 혹시 모를 0으로 나눠주는 상황을 피하기위해서이다. 이렇게 되면, 예를 들어 첫 번째 센서만 라인을 인식하게 되면 add_value에 -30이 저장되어, m1은 255, m2는 195의 속력으로 좌회전하며 주행하게 된다(m1이 오른쪽 모터이다). 또한, 두 번째, 세 번째 센서가 인식하면 add_value에는 0이 저장, 두 모터는 255의 속력으로 직진한다.</p> <p>기준속력을 225로 정한 데에는 이 값이 오버플로우를 막으면서 가장 큰 속력을 낼 수 있는 값이 225이기 때문이다. add_value의 최대/최소값은 각각 30/-30이고, 모터 PWM은 255를 초과하면 오버플로우가 일어나 다시 0으로 돌아가므로, 오버플로우가 일어나지 않는 기준속력의 최대값은 225이다.</p>
<pre>if(sensor1_v+sensor2_v+sensor3_v+sensor4_v==0) { if(add_value<0) { digitalWrite(e1,LOW); digitalWrite(e2,LOW); analogWrite(m1, 225-add_value); analogWrite(m2, 50); } if(add_value>0) { digitalWrite(e1,HIGH); digitalWrite(e2,HIGH); analogWrite(m1, 50); analogWrite(m2, 225+add_value); } goto re; }</pre>	<p>왼쪽의 코드는 라인트레이서의 안정성을 확보하기 위한 보정코드이다. 아래와 같은 상황을 상상해보자. 라인트레이서가 급커브를 만나 회전하다, 라인의 회전각도가 아두이노에 저장된 급격해 커브를 도는 중에 IR센서들이 라인 밖으로 나가버린 상황이다.</p> 

	<p>이 상황에서, 빠르게 회전하는 모터는 그대로 두되, 반대쪽 모터에 50만㎞의 역회전을 가하여 더 빠르게 라인으로 복귀하도록 코딩하였다. 또한 goto문을 이용하여 센서가 라인 밖으로 나갔을 때에는 원래의 코드(IR센서가 라인 위에 있을 때의 코드)를 실행하지 않고 바로 loop 맨 처음으로 돌아가도록 프로그래밍하였다. 이것이 이 코드가 코드 전문에서 원래의 코드보다 위에 있고, loop 첫 시작에 re: 가 있는 이유이다.</p>
<pre>if(sensor1_v+sensor2_v+sensor3_v+sensor4_v==4) { state=0; }</pre>	<p>센서 4개 모두가 검은색 선에 있을 경우에는 라인트레이서가 멈추는 코드이다. 단순히 모터 PWM에 0을 할당했을 때 라인트레이서가 멈추었다가 다시 움직이는 상황이 발생했기에, 더 이상 다른 동작을 할 수 없도록 state를 0으로 만들어 주었다.</p>

2. 메인 코드의 변화

맨 처음으로 시도한 코드는 if문을 사용한 코드였다. if문을 사용하여 왼쪽 센서가 라인을 인식하면 우회전을, 오른쪽 센서가 라인을 인식하면 좌회전을 하도록 코딩하였다. 다음 장의 코드가 if문을 사용한 코드로, loop내의 내용만을 나타내고 나머지 부분은 생략하였다. 그러나 이 코드의 경우 직선주로에서도 좌우로 크게 흔들렸고, 안정성 면에서도 뛰어나지 못했다. 완만한 커브의 경우 잘 통과했지만, 조금만 급격해져도 라인 밖으로 나가버리기 일쑤였다. LED를 이용해 디버깅해본 결과, 안전한 주행을 위해 가장 핵심적인 포인트는 ‘코너링에서 4개의 센서 모두가 라인 밖으로 나갔을 때의 대처’였다. 메모리에 센서가 라인 밖으로 나가기 이전 값을 저장하고, 센서가 나가면 회전을 더 강하게 하도록 하는 보정코드를 사용해보았지만, 라인트레이서의 안정성은 쉽사리 나아지지 않았다. 그 원인을 ‘loop가 한번 도는데 걸리는 시간’이라고 생각해 보았고, if 문을 사용하지 않는 현재의 코드, 즉 센서에 벨류를 할당하여 모터를 제어하는 ‘센서벨류코드’를 사용하게 되었다.

<pre> void loop() { sensor1_v=digitalRead(sensor1); sensor2_v=digitalRead(sensor2); sensor3_v=digitalRead(sensor3); sensor4_v=digitalRead(sensor4); //스위치를 이용해 라인트레이서의 상태를 전환하려면 이 코드를 사용한다. switch_value=digitalRead(switch_pin); if(switch_value==0)//스위치를 누르면 { if(switch_yes==1) { state = 1-state; switch_yes=0; } } if(switch_value==1)//스위치에서 손때면 { switch_yes=1; } if(state==1) { //직진, 빨간색LED if(sensor1_v==0&&sensor1_v==0&&sensor3_v==1&&sensor4_v==0) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,70); analogWrite(m2,70); m1memory=60; m2memory=60; led(1,0,0); } //우회전, 초록색LED if(sensor1_v==0&&sensor2_v==0&&sensor3_v==0&&sensor4_v==1) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,70); analogWrite(m2,0); m1memory=110; m2memory=0; led(0,1,0); } } if(sensor1_v==0&&sensor2_v==0&&sensor3_v==1&&sensor4_v==1) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,70); analogWrite(m2,0); m1memory=90; m2memory=0; led(0,1,0); } if(sensor1_v==0&&sensor2_v==0&&sensor3_v==1&&sensor4_v==0) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,70); analogWrite(m2,0); } } </pre>	<pre> m1memory=60; m2memory=60; led(1,0,0); } //좌회전, 노란색LED if(sensor1_v=1&&sensor2_v==0&&sensor3_v==0&&sensor4_v==0) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,0); analogWrite(m2,70); m1memory=0; m2memory=110; led(0,0,1); } if(sensor1_v==1&&sensor2_v==1&&sensor3_v==0&&sensor4_v==0) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,0); analogWrite(m2,70); m1memory=0; m2memory=90; led(0,0,1); } if(sensor1_v==0&&sensor2_v==1&&sensor3_v==0&&sensor4_v==0) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,0); analogWrite(m2,70); m1memory=60; m2memory=60; led(1,0,0); } //stop if(sensor1_v=1&&sensor2_v==1&&sensor3_v==1&&sensor4_v==1) { analogWrite(m1,0); analogWrite(m2,0); led(0,0,0); } //memory: 지나쳤을 때 if(sensor1_v==0&&sensor2_v==0&&sensor3_v==0&&sensor4_v==0) { digitalWrite(e1,HIGH); digitalWrite(e2,LOW); analogWrite(m1,m1memory); analogWrite(m2,m2memory); } } if(state==0) { analogWrite(m1,0); analogWrite(m2,0); led(0,0,0); } } </pre>
---	--

3. 보정 코드의 변화

센서벨류코드를 사용해도, 코너링에서 4개의 센서 모두가 라인 밖으로 나갔을 때의 대처는 잘 이루어지지 않았다. 이를 보정하기 위하여, 처음으로 사용해본 보정 코드는 다음과 같다.

```
if(sensor1_v+sensor2_v+sensor3_v+sensor4_v==0)
{
    digitalWrite(e1,LOW);
    digitalWrite(e2,HIGH);
    analogWrite(m1,255-(add_value+40));
    analogWrite(m2,225+(add_value+40));
    goto re;
}
```

그러나 위의 보정코드를 사용하여도, 회전을 너무 크게하여 IR센서가 건너편의 라인을 인식하는 일이 계속해서 일어났다. 이는 급커브에서도 안쪽 바퀴가 계속해서 앞쪽으로 돌기 때문이라고 판단하였고, 다음과 같은 보정코드를 작성하였다.

```
if(sensor1_v+sensor2_v+sensor3_v+sensor4_v==0)
{
    if(add_value<0)
    {
        digitalWrite(e1,LOW);
        digitalWrite(e2,HIGH);
        analogWrite(m1, 255-add_value);
        analogWrite(m2, 0);
    }
    if(add_value>0)
    {
        digitalWrite(e1,LOW);
        digitalWrite(e2,HIGH);
        analogWrite(m1, 0);
        analogWrite(m2, 255+add_value);
    }
    goto re;
}
```

위 코드는 급격한 우회전 시에는 오른쪽 모터의 출력을 0으로, 급격한 좌회전 시에는 왼쪽 모터의 출력을 0으로 만드는 보정코드이다. 그러나 위 코드를 사용해도 차체의 안정성은 아무 미비하게 증가할 뿐, 여전히 급커브에서는 제대로 작동하지 않았다. 급커브에서 조금 더 빠르고 정확하게 회전하기 위해, 급회전 시에 안쪽바퀴에 역회전을 걸어보기로 하였다.

```

f(sensor1_v+sensor2_v+sensor3_v+sensor4_v==0)
{
    if(add_value<0)
    {
        digitalWrite(e1,LOW);
        digitalWrite(e2,LOW);
        analogWrite(m1, 225-add_value);
        analogWrite(m2, 50);
    }
    if(add_value>0)
    {
        digitalWrite(e1,HIGH);
        digitalWrite(e2,HIGH);
        analogWrite(m1, 50);
        analogWrite(m2, 225+add_value);
    }
    goto re;
}

```

이렇게 하여 만들어진 것이 코드 전문 및 설명에서 설명한 현재의 보정코드이다. 안쪽바퀴에 역회전을 50으로 주니 급회전 구간도 매끄럽게 통과했고, 모든 구간에서의 안정성이 크게 향상되었다.

IV. 한계점 및 해결방안

한계점으로는 공기저항이나 하드보드지의 무게를 전혀 고려하지 않은 외관을 들 수 있다. 앞쪽에 솟아난 눈은 큰 공기저항을 만들어냈을 것이다. 또한, 모터와 모터사이의 거리, 그리고 센서와 모터사이의 거리를 크게 고려하지 않고 기초골격을 디자인하였다는 점이다.

개선방안으로, 먼저 외관을 조금 더 유선형에 가깝게 디자인한다면 공기의 저항을 최소화 시킬 수 있을 것이다. 또한, 하드보드지가 아닌 조금 더 가벼운 재료를 사용해 외관을 꾸민다면 바퀴와 바닥사이의 마찰력이 줄어 더 빠른 속력을 낼 수 있을 것이다. 마지막으로 같은 환경에서 모터와 모터사이의 거리, 센서와 모터사이의 거리를 각각 바꿔보며 시험주행을 해본다면 적절한 거리 값을 도출해 낼 수 있을 것이다.