

GY

You

영남대학교 자율주행 동아리 매니악을 알아?



ChatGPT

네, 알겠습니다. 영남대학교의 자율주행 동아리인 "매니악"에 대해 알려드리겠습니다. "매니악"은 영남대학교에서 활동하는 자율주행 분야의 학술 동아리 중 하나입니다. 이 동아리는 자동차 및 로봇공학, 인공지능 등과 관련된 기술 및 이론에 대해 연구하고, 이를 실제 프로젝트에 응용하여 자율주행 기술을 개발하고 있습니다. "매니악"의 회원들은 함께 공부하고 아이디어를 공유하며 다양한 프로젝트를 수행하며 자율주행 분야에서의 경험을 쌓고 있을 것입니다. 이 동아리에 대해 더 자세한 정보를 원하신다면 해당 동아리의 공식 웹사이트나 소셜 미디어 페이지를 방문하시거나 직접 동아리 멤버들에게 문의하시는 것이 좋습니다.



즐겁다



# 매니악 봄은 온다



GPS 부원이  
되고 싶은 자는  
나에게...

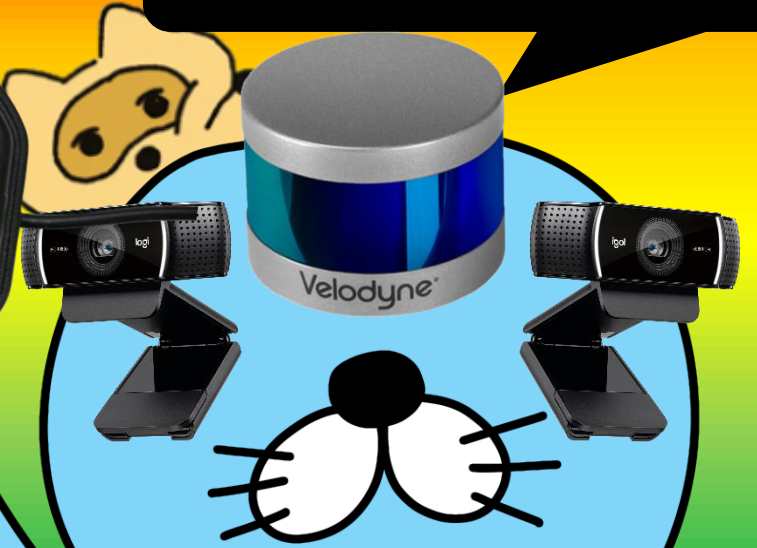
♡ GPS 많관부~~ ♡

집중! 경청해 주세요~!

신입 부원 +  
종합  강 + 월

불평 금지

알아두면 적당히 아는 척 하기 좋습니다.  
4학년이 아는 대로 적은 겁니다. 맹신 하십쇼.



**1. 차량 제어**

**2. 리모~~(물고기 아님 ㅋㅋ)**



이야기 전에...

이 자료는 전문적인 지식보다

실전에 사용하기 위해 경험을 바탕으로 (전국 7등ㅋㅋ) 압축되고 각색된 내용이 많습니다.

그리고 저 ppt 잘 못 만듭니다.

그치만 야매 강의거나 오개념 전파는 아니니 걱정마시길...

그럼 실전 길거리 자율주행 이야기 시작합니다~~~

(이해 안되거나 잘 못 났다 싶은건 바로 말해주세요영)

# 01 차량 제어

차를 움직일 때 필요한 값은 뭐가 있을까?



(삼각함수 모르면 참고하세요)

간단히 차와 가고 싶은 좌표의 각도를 삼각 함수로 구해도 차는 굴러간다.

그치만 이왕 하는 자율주행 좀 더 **쌈@뽕**하게 조향 각을 구하는 방법이 있는데  
그게 바로 **Pure pursuit**이라는 알고리즘이다.

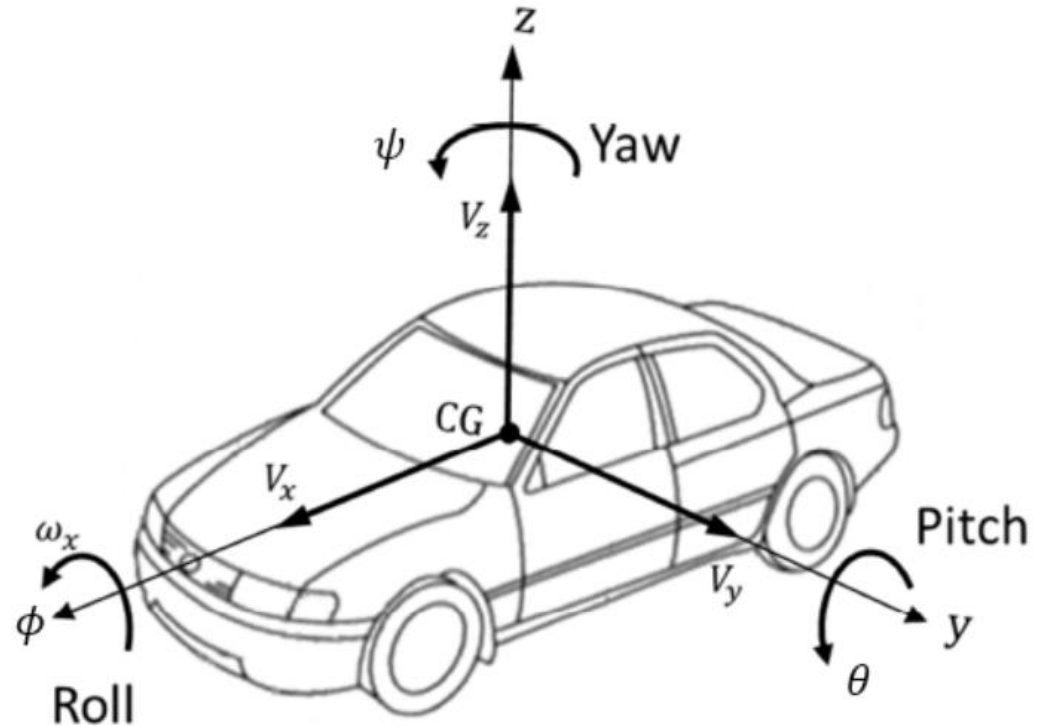
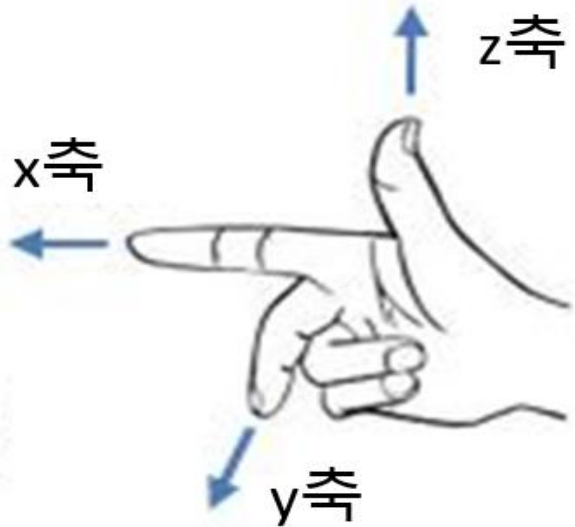
# 01 차량 제어

조향 각을 구하기 전에

차량에 대한 좌표와 축에 대해 알고 있어야 합니다. (좌표 중요함  $\approx \circ$ )

ISO의 차량 좌표 축 기준)

차량의 정면 +X축, 오른쪽 -Y, 왼쪽 +Y



# 01 차량 제어

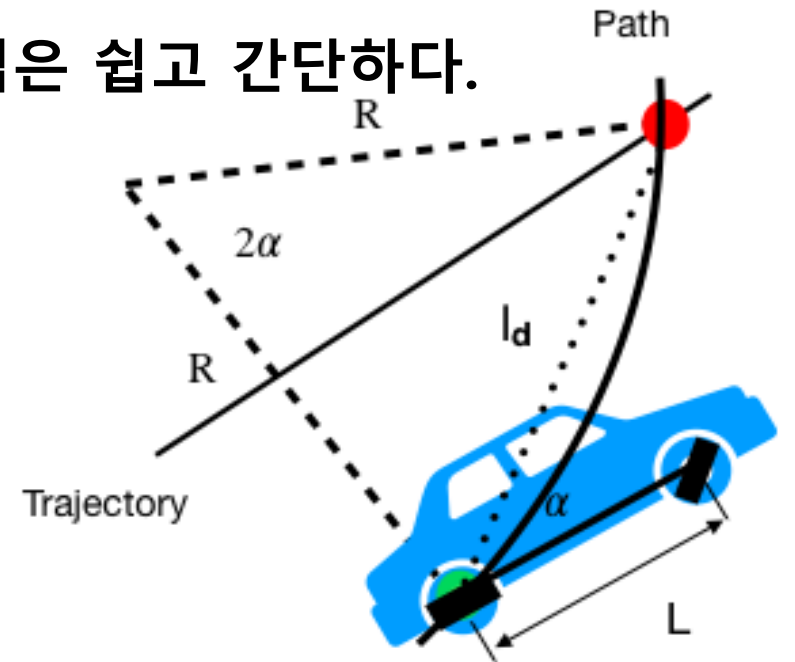
1. `catkin_make && source devel/setup.bash`
2. `simul_car.cpp` 실행
3. `rviz` 실행 `$ rosrun rviz rviz`
4. `rviz` 화면에서 `Ctrl + O`를 눌러 `welcome_drink`패키지의 `rviz`폴더 안에 `edu_rviz` 실행

# 01 차량 제어\_Pure pursuit

GPS를 통해 내 차의 좌표를 알 수 있으니  
단순히 다른 좌표와의 각도 차이를 삼각함수로 구하는 법은 쉽고 간단하다.

하지만 Pure pursuit의 **쌈@뽕**한 점은  
차의 뒷바퀴를 기준으로 각도를 구한다는 것이다.  
(뒷바퀴가 기준이라는건 뒷바퀴가 점에 도착하기 위한 조향각을 구한다는 뜻)

혹시나 너무 어렵다면  
어렵게 생각하지 말고 그냥 조향각 계산해주는 공식이구나 하고 넘어가자





# 01 차량 제어\_Pure pursuit

그리고 이걸 전공 수업이 아니기에  
이 자료에선 pure pursuit의 딱딱한 공식 이론 말고  
이 녀석의 느낌과 실전 이용법을 배우려 한다.

Look Ahead distance(줄여서 look\_d) 이거 하나만 알고 가자.  
단어 뜻 그대로 차량이 바라보는 거리를 뜻한다.

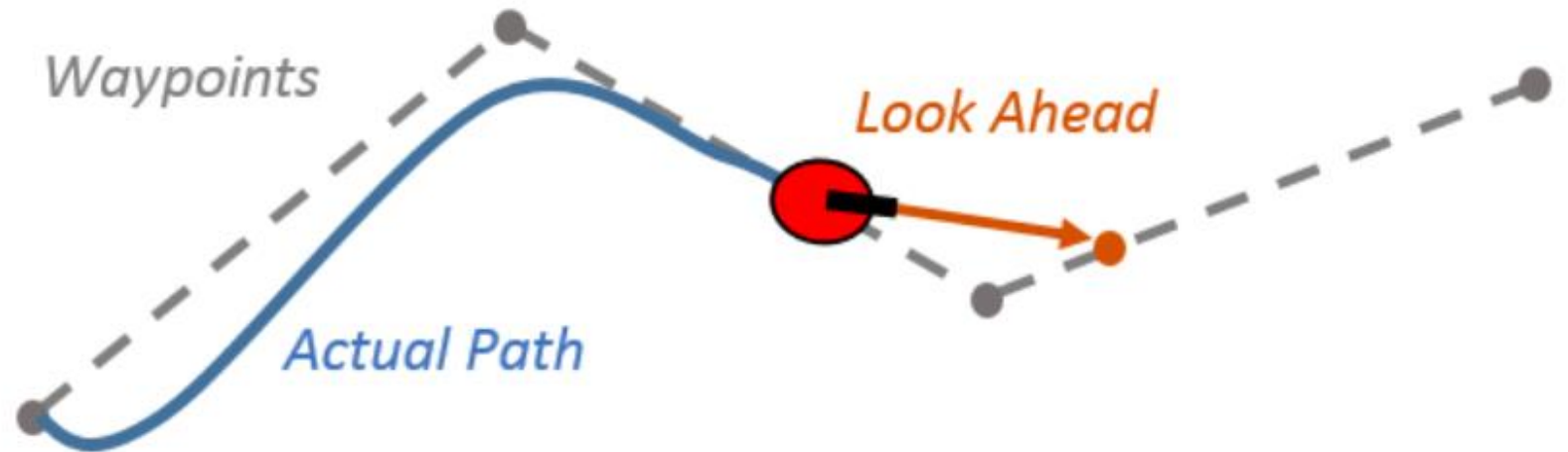
자세한 내용은  
뒷장 더보기...

$$\frac{ld}{\sin 2\alpha} = \frac{R}{\sin(\frac{\pi}{2} - \alpha)}$$

$$\frac{ld}{2\sin\alpha\cos\alpha} = \frac{R}{\cos(\alpha)}$$

$$\frac{ld}{\sin\alpha} = 2R$$

$$k = \frac{1}{R} = \frac{2\sin\alpha}{ld}$$



# 01 차량 제어\_Pure pursuit

Look Ahead Distance가 왜 중요하냐

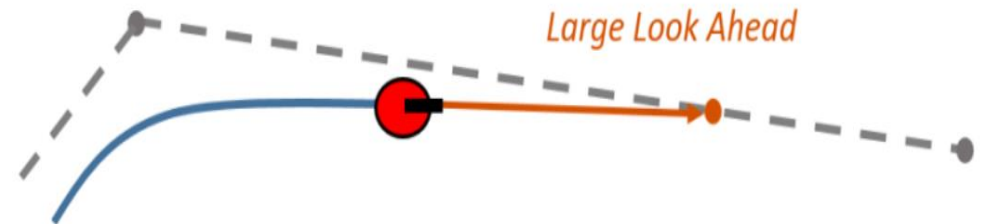
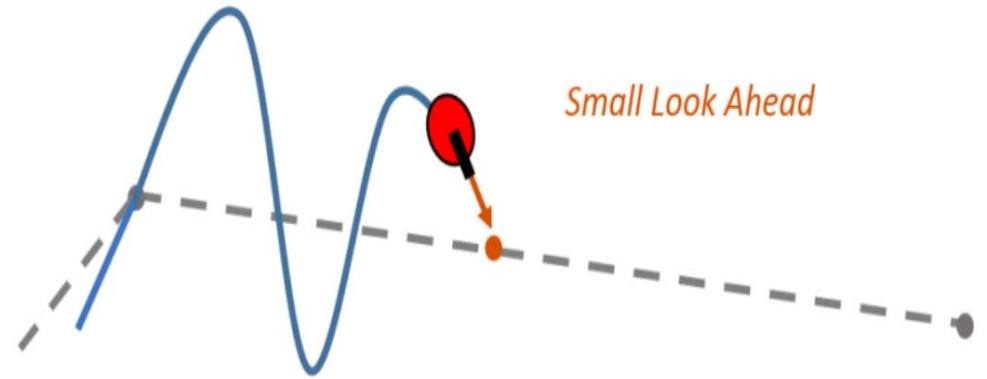
이 친구 하나로 차가 움직이는 궤적이 달라진다.

점과 점 사이 각도로 하는 조향과

Pure pursuit으로 하는 조향의 차이점,

Look\_d가 왜 중요한지, 왜 궤적이 달라지는지

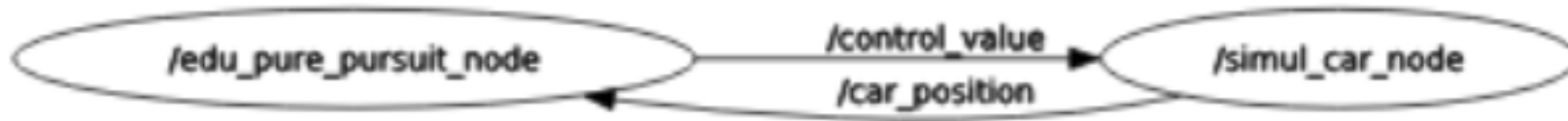
코드로 직접 알아보고 실행해보자.



# 01 차량 제어

시뮬레이션 하기 전에!

데이터의 흐름을 읽을 줄 알아야합니다(from rqt\_graph)



```
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/Point.h>
```

>> 로봇의 속도들을 나타내기 적합함

>> 점의 좌표점을 나타내기 적합함

# 01 차량 제어

## geometry\_msgs/Twist의 구조

```
---  
linear:  
  x: 0.3  
  y: 0.0  
  z: 0.0  
angular:  
  x: 0.0  
  y: 0.0  
  z: -2.402214765548706  
---
```

## geometry\_msgs/Point의 구조

```
---  
x: 5.000901768691094  
y: 2.767568928389771  
z: 0.0  
---
```

# 01 차량 제어

```
24 float Calculate_steering()
25 {
26     double dx = (target_point.first - car_pos_x);
27     double dy = (target_point.second - car_pos_y);
28
29     double dist = sqrt(dx*dx + dy*dy);
30
31     double alpha = atan2(dy, dx);
32
33     float steering_rad = atan2((2*1.04*sin(alpha)), look_distance);
34
35     return steering_rad;
36 }

16 //////////////////////////////////////////////////
17 std::pair<double ,double> target_point = {5.0, 2.76756905};
18
19 double car_velocity = 0.3;
20
21 double look_distance = 3.0;
22 //////////////////////////////////////////////////

44 void Publish_control_value()
45 {
46     geometry_msgs::Twist car_control;
47
48     car_control.linear.x = car_velocity;
49     car_control.angular.z = car_steering_rad;
50
51     car_control_pub.publish(car_control);
52 }
```

# 01 차량 제어

결론)

차량의 조향 각은 삼각함수로 구하든, pure pursuit으로 구하든 차는 굴러갑니다.

하지만

점과 점사이를 최단 거리로 가는 삼각함수 조향을 사용할지

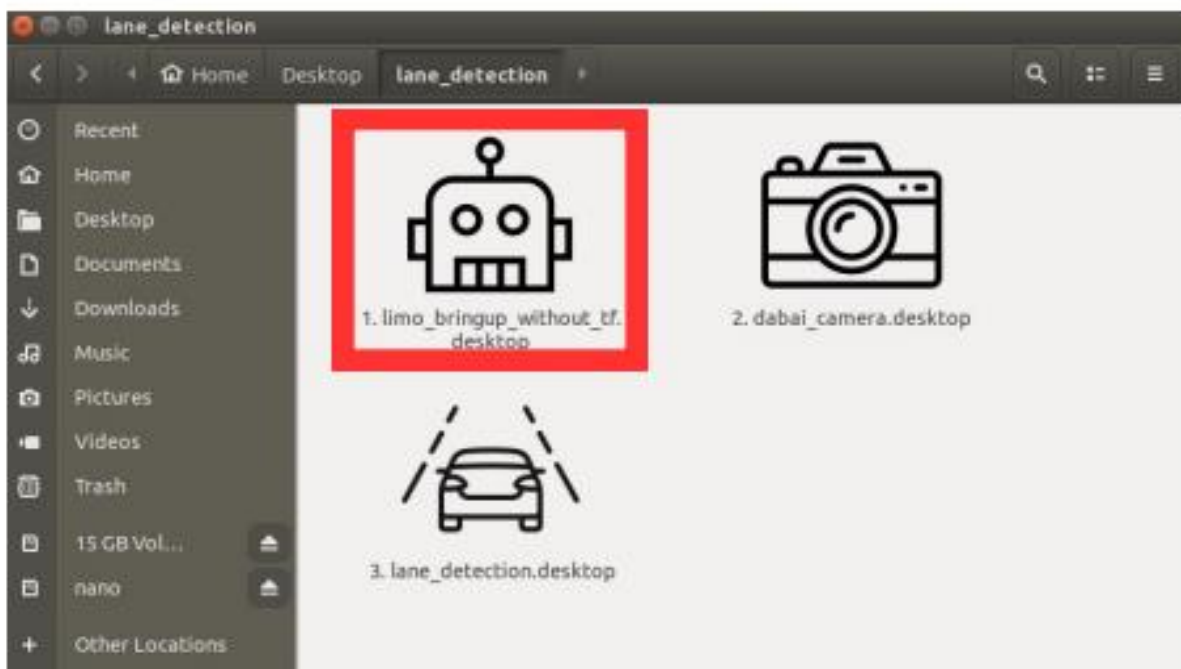
점이 존재하는 궤적을 따라가는 pure pursuit 조향을 사용할지

두개의 차이점을 기억하고 상황에 더 적합한 방법 사용 기억해주기~~~

# ROS BRING UP 켜고 끄는 방법!

## 켜기

파일 탐색기 : HOME -> Desktop -> lane\_detection  
-> 1. limo\_bringup\_without\_tf.desktop



## 끄기

#terminal : Ctrl + c 를 눌러 bring up 종료



## 02 리모~

ROS 토픽은 여러가지 정보를 담고 있음.

2D 라이다의 경우

angle\_min, angle\_max, angle\_increment, range\_min, range\_max, ranges 가 중요함.

```
gyunddo@gyunddo:~$ rostopic echo /scan
header:
  seq: 1625
  stamp:
    secs: 1714645849
    nsecs: 867963000
  frame_id: "laser_link"
angle_min: -1.5707963705062866
angle_max: 1.5707963705062866
angle_increment: 0.006996865849941969
time_increment: 0.0002500000118743628
scan_time: 0.11174999922513962
range_min: 0.10000000149011612
range_max: 12.0
ranges: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4.4724998474121
```

angle\_min >> 인식하는 최소 각도[rad]

angle\_max >> 인식하는 최대 각도[rad]

angle\_increment

>> 인식하는 각도 상승 폭[rad]

range\_min >> 인식하는 최소 거리[m]

range\_max >> 인식하는 최대 거리[m]

ranges >> 인식한 포인트들의 거리 값들[m]



## 02 리모~

```
6   int sum_point = 0;
7   double roi_distance = 1.5;    //unit : m
```

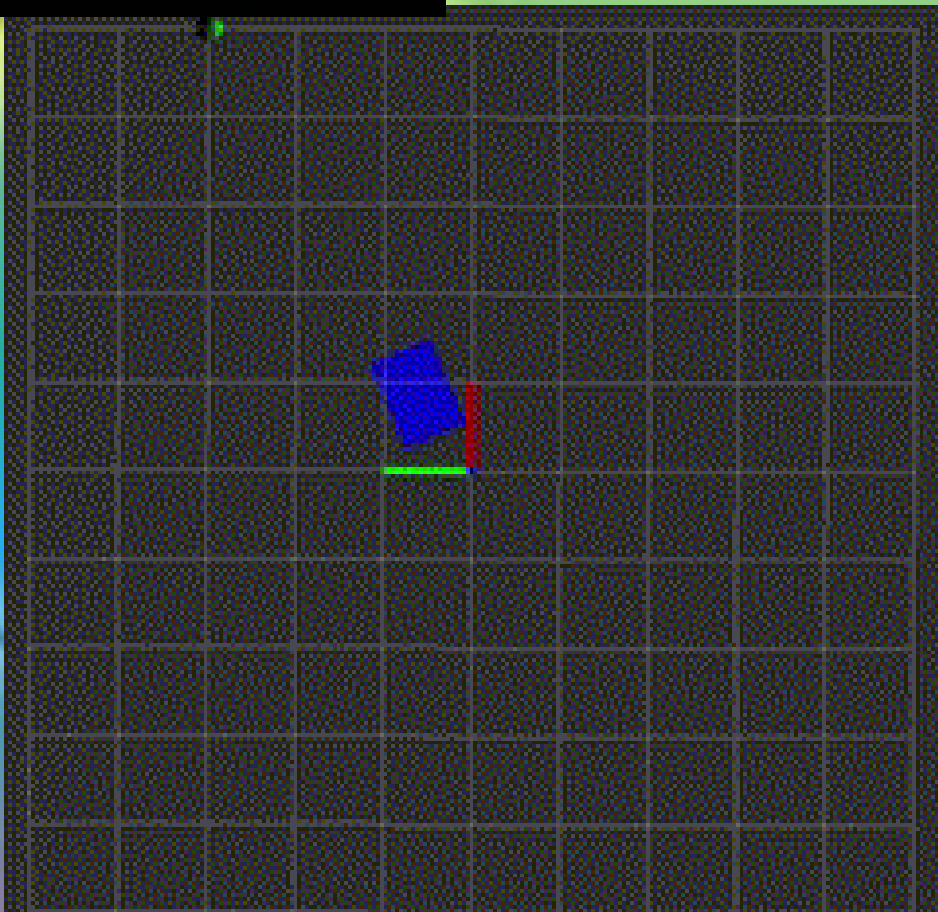
```
21 void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan)
22 {
23     int count = scan->scan_time / scan->time_increment;
24     //int count=(int)(360./RAD2DEG(scan->angle_increment));
25
26     sum_point = 0;
27
28     for(int i=0; i<count; i++)
29     {
30         double point_rad = scan->angle_min + scan->angle_increment * i;
31         double point_deg = Rad2Deg(point_rad);
32
33         if((point_deg >= -1*30)&&(point_deg <= 30))
34         {
35             if((scan->ranges[i] <= roi_distance)&&(scan->ranges[i] >= scan->range_min))
36             {
37                 sum_point++;
38
39                 double point_x = scan->ranges[i]*cos(point_rad);
40                 double point_y = scan->ranges[i]*sin(point_rad);
41
42                 cout << sum_point << " : [X : " << point_x << "m, Y : " << point_y << "m]" << endl;
43             }
44         }
45     }
46 }
```



# ★ 숙제 ★



<Pure pursuit을 이용해서 Target point에서 멈추기>



1) Target point (5.0, 3.0)

2) Target point에서  
속도가 0이 되어야함

1등한 사람 재형 선배가 커피 사줌ㅋㅋ  
기한 : 5월 9일까지

# ★ QnA ★

(공격적인 질문 무서워요..ㅠㅠ)



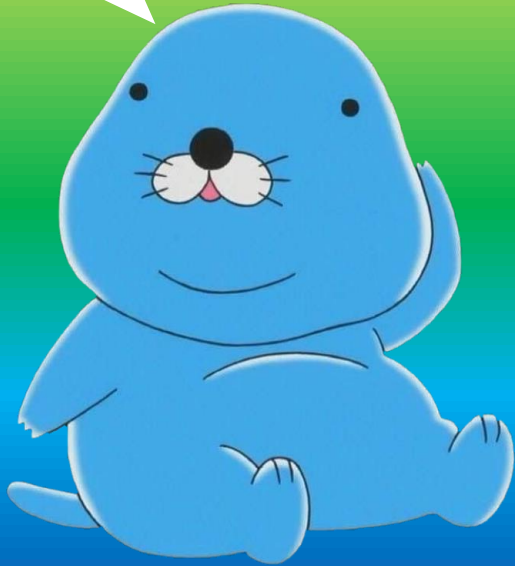


3D라이다 궁금하면  
라이다 강의 자료도 있습니다

# 감사합니다!!

## 좋은 하루 보내세요~!

먼 소린지 하나도  
모르면 찾아오세요



♥ 고생 많았어요 뽕뽕뽕 ♥



GPS 부원이  
되고 싶은 자는  
나에게...

캐릭터 출처 :  
귀여운 보농보농