

MORAI

경로 생성 및 종, 횡 방향 제어

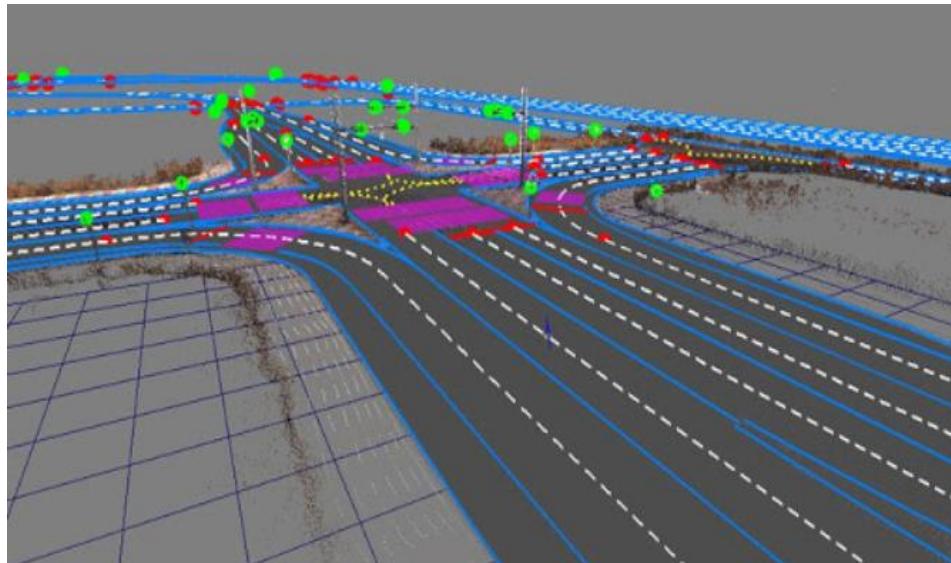
2021-04-08

경로

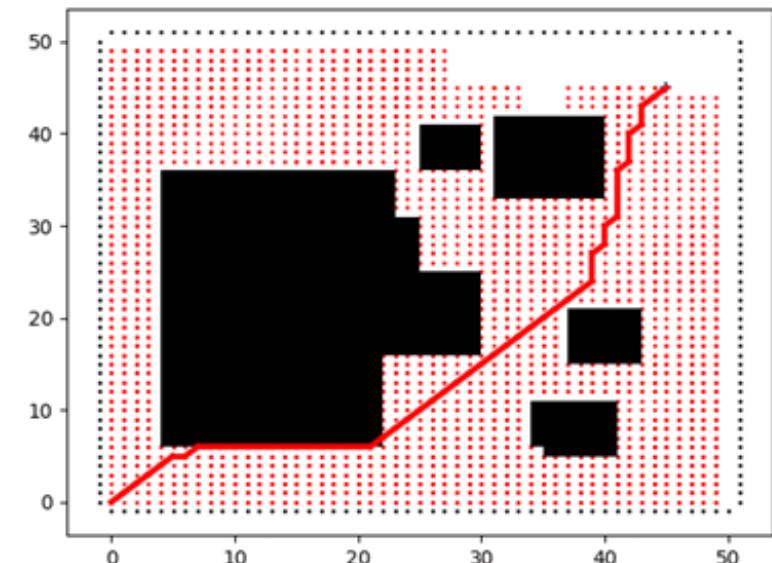
경로

▶ 경로란?

- 경로는 로봇이 존재할 수 있는 위치의 집합이다.
- 즉, 로봇이 가야할 길이다.
- 경로를 표현하는 방법에는 점으로 표현하는 방법과 그리드로 표현하는 방법이 있다.
- 경로를 만들 때는 안전하고 효율적인(최단) 경로를 만들어야 한다.



☞ 정밀도로지도 (점)

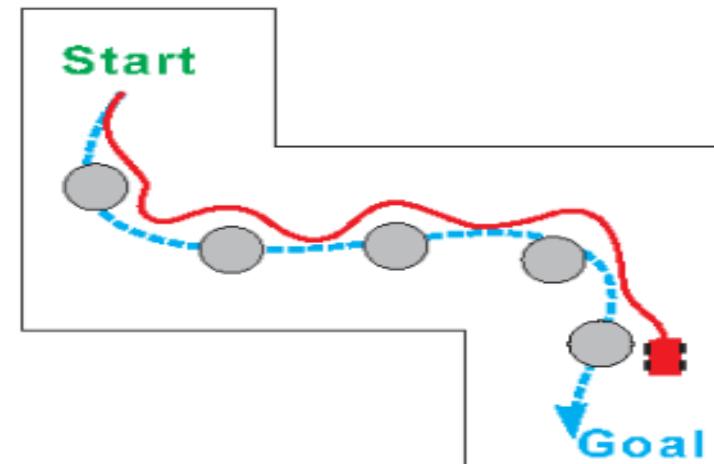
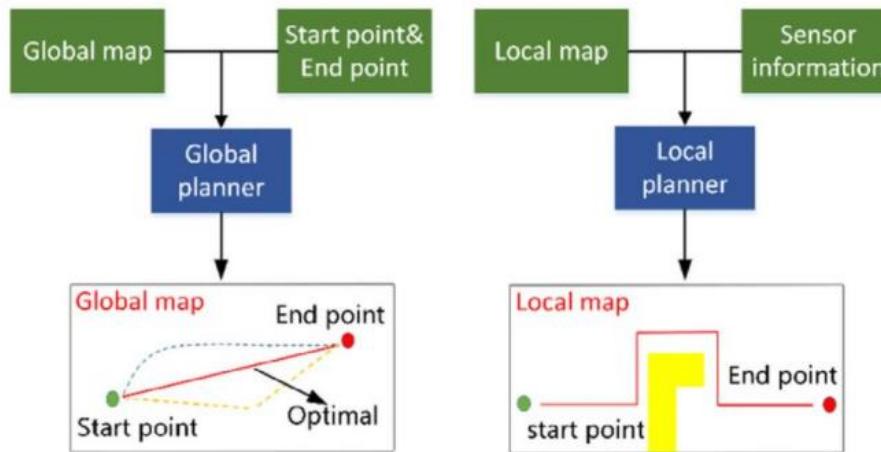


☞ 그리드 맵 (그리드)

경로

➤ 전역 경로와 지역 경로

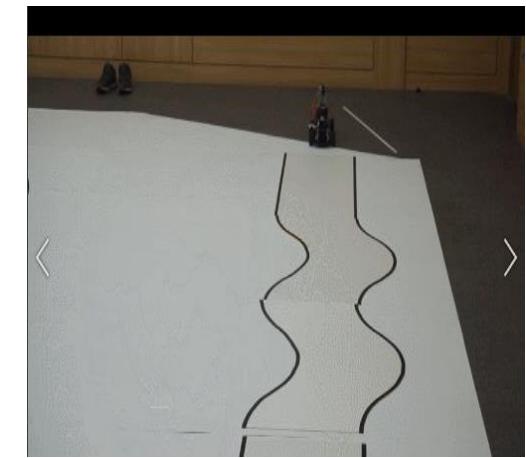
- 전역 경로
 - ✓ 출발 위치에서 목적지까지를 연결하는 경로
 - ✓ 예) 네비게이션 어플
- 지역 경로
 - ✓ 실제 로봇이 주행하는 경로
 - ✓ 예) 충돌 회피



경로

➤ 실내 로봇의 경로

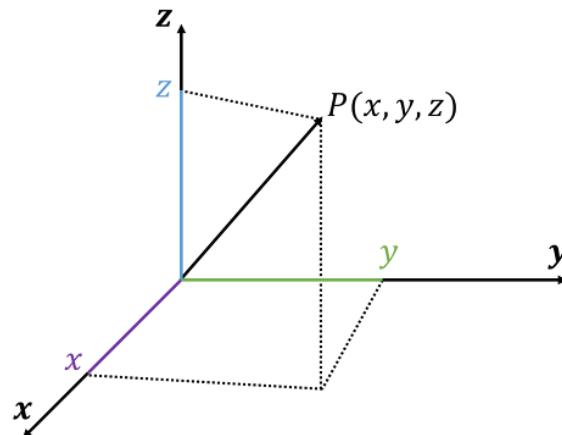
- 전역경로 사용 로봇
 - ✓ 라인트레이서
 - ✓ 아마존 물류로봇 키바
- 지역경로 사용 로봇
 - ✓ 지능형 모형차



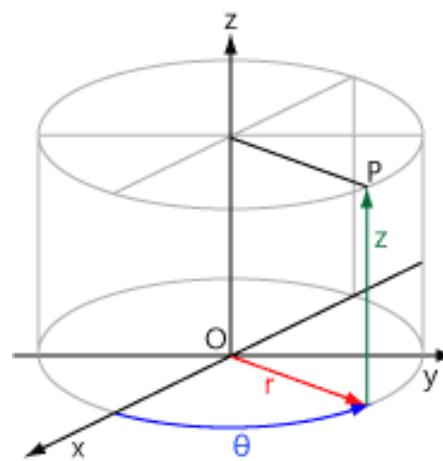
좌표계

- 공간상에서 물체의 위치를 표현하는 체계

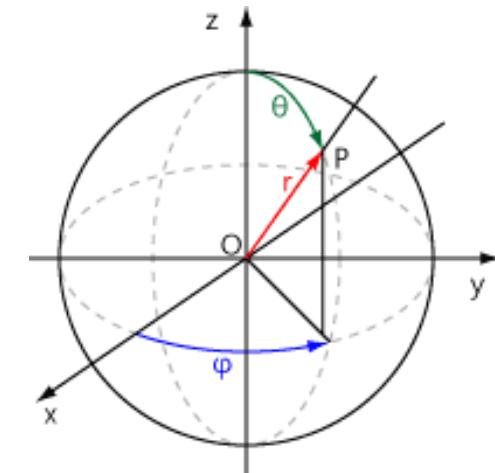
- ✓ 직교 좌표계, 원통 좌표계, 구면 좌표계 등 목적에 따라 여러 좌표계로 물체의 위치와 자세를 표현할 수 있다.
- ✓ 센서들은 각각 다른 좌표계를 사용한다.



직교 좌표계(IMU)



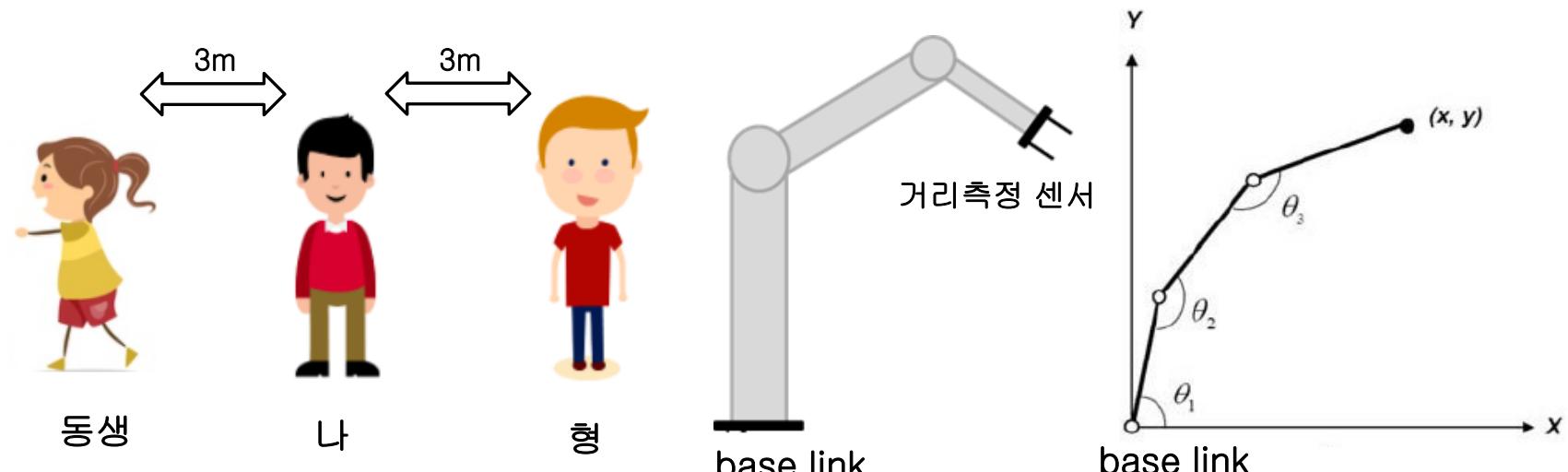
원통 좌표계(라이다)



구면 좌표계(GPS)

▶ 기준 좌표계

- 물체의 위치를 나타낼 때 사용하는 좌표계를 기준 좌표계라고 한다.
 - ✓ 나를 기준으로 형을 나타낸 표현과 동생을 기준으로 형을 나타낸 표현은 다르다.
 - 나 기준의 형 : 우측으로 3m, 동생 기준의 형 : 우측으로 6m
 - ✓ 어떤 기준으로 물체의 위치를 표현하느냐에 따라서 같은 위치의 물체라도 다르게 표현될 수 있다.
- 로봇에서는 하나의 좌표계가 아니라 여러 개의 좌표계를 사용한다.

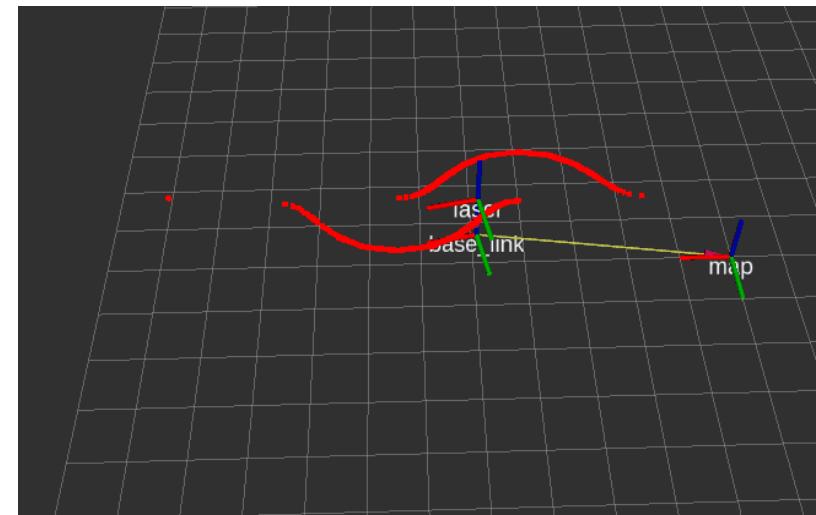
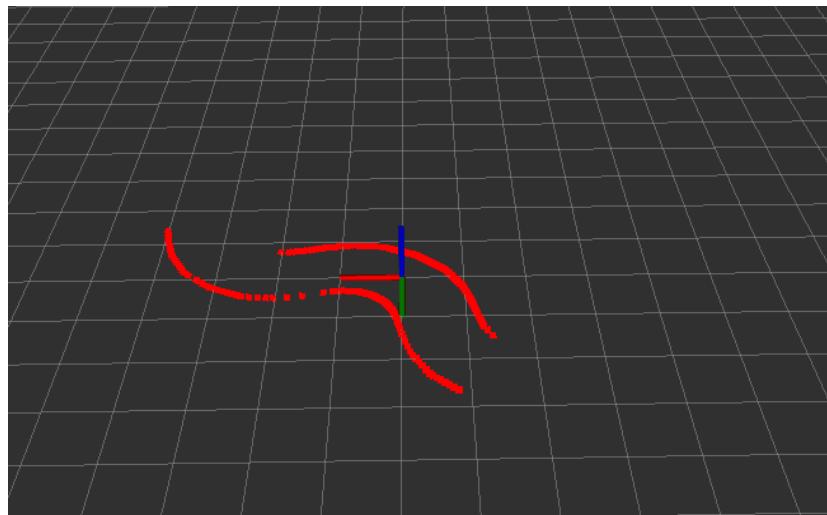


☞ 기준을 어디에 두느냐에 따라 위치가 다르게 표현된다. ☞ 거리측정 센서에서 측정한 거리는 base link 기준으로 표현되어어야 한다.

경로

➤ 기준 좌표계에 따른 데이터 표현 방법

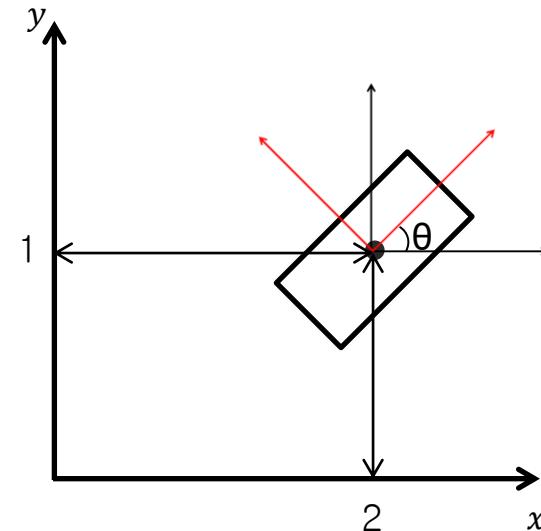
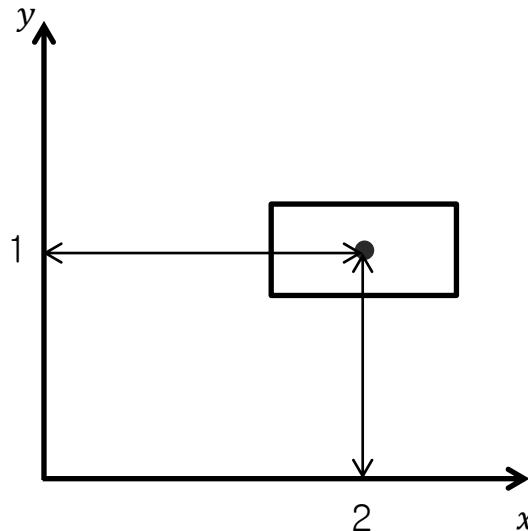
- map(아래 우측 영상)을 기준 좌표계로 데이터를 표현해야 전체 맵을 이해하기 쉽다.
- RVIZ는 시각화 툴로 ROS 좌표계(TF) 기능을 사용하면 원하는 좌표계를 기준으로 놓고 데이터를 볼 수 있다.



☞ 라이다 좌표계에서 본 라이다 데이터(좌), map 좌표계에서 본 라이다 데이터(우)

➤ 물체의 자세란?

- 기준 좌표계에서 물체가 얼마나 회전되어 있는지를 나타낸다.
 - ✓ 같은 위치라도 다른 자세를 취할 수 있다.
 - ✓ 표현방법은 오일러 각, 쿼터니언 두 가지가 있다.



☞ 같은 위치라도 물체의 자세는 다를 수 있다.

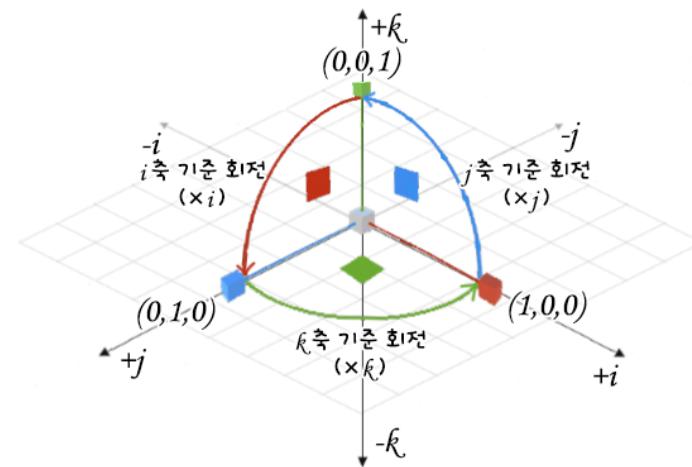
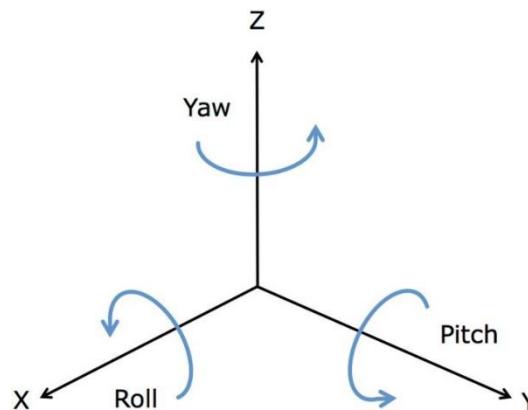
경로

➤ 오일러각

- 물체의 자세를 Roll Pitch Yaw 3가지로 표현하는 방법
- 직관적으로 이해하기 쉽다.
- 몇몇 자세를 표현하지 못하는 짐벌락 현상이 있다.

➤ 쿼터니언

- 물체의 자세를 x y z w 4가지 변수로 표현하는 방법
- 직관적으로 판별하기 어렵다.
- 모든 자세를 표현할 수 있으며 빠른 연산처리가 가능하여 컴퓨터 그래픽에서 주로 사용하는 좌표계다.



좌표계에서의 오일러각 표현 방식(좌)과 쿼터니언 표현 방식(우)

➤ TF

- ROS에서 제공하는 좌표계 관련 패키지
- 좌표계간 상관관계를 정의한다.
- 정의된 좌표계간에는 좌표변환을 지원한다.
- 오일러각 <-> 쿼터니언 데이터 변환을 해주는 기능이 있다.
 - ✓ `tf.transformations.quaternion_from_euler(roll,pitch,yaw)`
 - ✓ `tf.transformations.euler_from_quaternion(x,y,z,w)`
- 고정좌표계 사용
 - ✓ `rosrun tf static_transform_publisher x y z R P Y /frame_id /child_frame_id periods`
- TF 튜토리얼

<http://wiki.ros.org/tf/Tutorials>

$$\mathbf{q}_{IB} = \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \\ 0 \end{bmatrix} \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} \cos(\phi/2) \cos(\theta/2) \cos(\psi/2) + \sin(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \sin(\phi/2) \cos(\theta/2) \cos(\psi/2) - \cos(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \sin(\theta/2) \cos(\psi/2) + \sin(\phi/2) \cos(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \sin(\theta/2) \sin(\psi/2) - \sin(\phi/2) \sin(\theta/2) \cos(\psi/2) \end{bmatrix}$$

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \text{asin}(2(q_0 q_2 - q_3 q_1)) \\ \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

경로

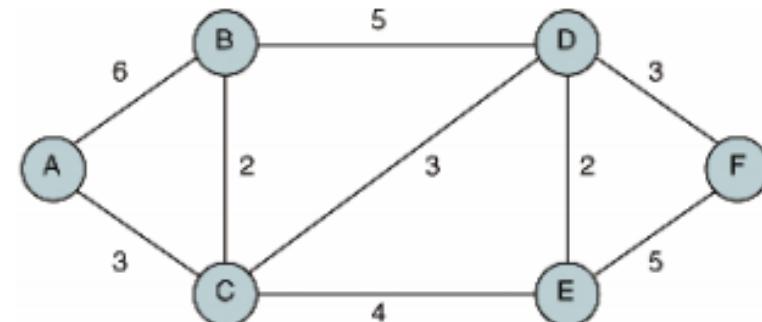
▶ 전역 경로 생성

• 정적 경로 생성

- ✓ 미리 경로를 직접 그리거나, 주행한 경로를 로깅해서 생성
- ✓ 한번 만들어진 경로는 출발지, 목적지가 항상 같다.
- ✓ 경로를 만드는 가장 쉬운 방법이나, 제약사항이 많다.

• 동적 경로 생성

- ✓ 맵 데이터 기반으로 경로탐색알고리즘을 이용해 동적으로 경로 생성
- ✓ Dijkstra, A*와 같은 알고리즘을 사용
- ✓ 맵 데이터를 구축하고, 경로탐색 알고리즘을 구현해야하는 번거로움이 있으나, 목적지만 찍으면 출발지로 부터 목적지까지 가능 경로를 만들어 줄 수 있다.



➤ 실습 1 : 전역경로 생성

- 센서 데이터(GPS, IMU)를 이용해 로봇의 Global Pose를 알 수 있다.
- 로봇의 위치를 기록한 점들의 집합은 경로가 된다.
- 경로를 생성해 텍스트파일로 저장하자.
 - 텍스트 파일에는 x, y, yaw(rad)을 저장하고, 데이터간 간격은 탭(\t)으로 구분하고, 매 순간 기록한 경로점 간 데이터는 줄 바꿈(\n)으로 구분한다.
- ROS 메시지는 nav_msgs/Path 타입을 사용한다.



1	22.2075939708	1113.5803212	0.826583825622
2	22.3162460867	1113.6995839	0.831919602426
3	22.4544239581	1113.852538	0.836109063731
4	22.6188889082	1114.03600968	0.839970902933
5	22.7925034107	1114.24633683	0.880728708604
6	22.9478073657	1114.4718007	0.967613383957
7	23.075334602	1114.70666398	1.07335487413
8	23.1771107265	1114.9594716	1.18806495169
9	23.2499008715	1115.22412003	1.30238857364
10	23.2945747914	1115.49951066	1.40997704668
11	23.3252010873	1115.80786026	1.47179773335
12	23.3539257589	1116.1070546	1.47508294424
13	23.3834839422	1116.41491593	1.4750783682
14	23.4135914373	1116.71215714	1.46985083392
15	23.4497719344	1116.97607316	1.43455464666
16	23.5117397841	1117.23596085	1.33672607693
17	23.6034374762	1117.4940175	1.22937274746
18	23.7205048149	1117.73779191	1.12309092874
19	23.8654633097	1117.96813859	1.00910671695
20	24.0329781112	1118.17968644	0.901042844436
21	24.2229652941	1118.37084855	0.788480739807

➤ 실습 1 : 전역경로 생성

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import rospy
5  import rospkg
6  from math import cos,sin,pi,sqrt,pow
7  from geometry_msgs.msg import PoseStamped
8  from nav_msgs.msg import Odometry,Path
9  import tf
10 ∵ class make_path :
11    ∵ def __init__(self):
12        rospy.init_node('make_path', anonymous=True)
13        rospy.Subscriber("odom", Odometry, self.odom_callback)
14        self.path_pub = rospy.Publisher('/path',Path, queue_size=1)
15        self.is_odom=False
16        self.path_msg=Path()
17        self.path_msg.header.frame_id='/map'
18        self.prev_x=0
19        self.prev_y=0
20        rospack=rospkg.RosPack()
21        pkg_path=rospack.get_path('av_alg_mgko')
22        full_path=pkg_path+'/path'+ '/test.txt'
23        self.f=open(full_path,'w')
24
25    ∵ while not rospy.is_shutdown():
26        |   rospy.spin()
27        self.f.close()
28
29    ∵ def odom_callback(self,msg):
30        waypoint_pose=PoseStamped()
31        x=msg.pose.pose.position.x
```

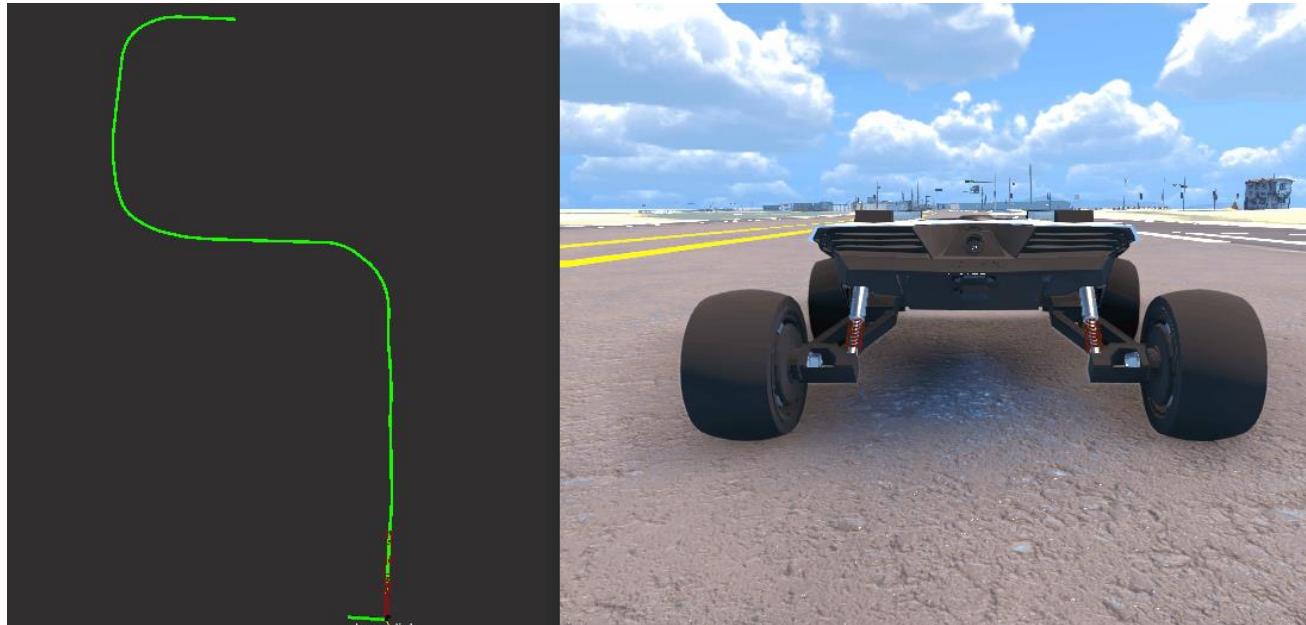
➤ 실습 1 : 전역경로 생성

```
29    def odom_callback(self,msg):
30        waypint_pose=PoseStamped()
31        x=msg.pose.pose.position.x
32        y=msg.pose.pose.position.y
33        odom_quaternion=(msg.pose.pose.orientation.x,msg.pose.pose.orientation.y,msg.pose.pose.orientation.z,msg.pose.pose.orientation.w)
34        yaw=tf.transformations.euler_from_quaternion(odom_quaternion)[2]
35    if self.is_odom== True :
36        distance=sqrt(pow(x-self.prev_x,2)+pow(y-self.prev_y,2))
37    if distance > 0.1 :
38        waypint_pose.pose.position.x=x
39        waypint_pose.pose.position.y=y
40        waypint_pose.pose.orientation.w=1
41        self.path_msg.poses.append(waypint_pose)
42        self.path_pub.publish(self.path_msg)
43        data='{0}\t{1}\t{2}\n'.format(x,y,yaw)
44        self.f.write(data)
45        self.prev_x=x
46        self.prev_y=y
47        print(x,y)
48
49
50    else :
51        self.is_odom=True
52        self.prev_x=x
53        self.prev_y=y
54
55
56
57 if __name__ == '__main__':
58 try:
59     test_track=make_path()
60 except rospy.ROSInterruptException:
61     pass
```

경로

➤ 실습 2 : 경로 읽어오기 및 지역경로 생성

- 저장한 텍스트 파일을 읽어서 경로 메시지에 담아서 전역경로를 Publish 한다.
- 전역경로의 경로점 중 로봇과 가장 가까운 경로점을 탐색한다.
- 가장 가까운 경로점을 시작으로하는 지역경로를 생성해서 메시지로 Publish 한다.
 - 원하는 지역경로의 길이만큼 경로점을 가져온다.



☞ 결과 영상

초록선(전역경로), 빨간선(지역경로)

➤ 실습 2 : 경로 읽어오기 및 지역경로 생성

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import rospy
5  import rospkg
6  from math import cos,sin,pi,sqrt,pow
7  from geometry_msgs.msg import PoseStamped
8  from nav_msgs.msg import Odometry,Path
9
10
11 class path_pub_tf :
12
13     def __init__(self):
14         rospy.init_node('path_pub', anonymous=True)
15         rospy.Subscriber("odom", Odometry, self.odom_callback)
16         self.global_path_pub = rospy.Publisher('/global_path',Path, queue_size=1)
17         self.local_path_pub = rospy.Publisher('/local_path',Path, queue_size=1)
18         self.global_path_msg=Path()
19         self.global_path_msg.header.frame_id='/map'
20
21         self.is_odom=False
22         self.local_path_size=50
23
24         rospack=rospkg.RosPack()
25         pkg_path=rospack.get_path('av_alg_mgko')
26         full_path=pkg_path+'/'+path+'/'+test.txt'
27         self.f=open(full_path,'r')
28         Lines=self.f.readlines()
29
30         for line in lines :
31             tmp=line.split()
32             read_pose=PoseStamped()
33             read_pose.pose.position.x=float(tmp[0])
34             read_pose.pose.position.y=float(tmp[1])
35             read_pose.pose.orientation.w=1
36             self.global_path_msg.poses.append(read_pose)
```

➤ 실습 2 : 경로 읽어오기 및 지역경로 생성

```
35         self.global_path_msg.poses.append(read_pose)
36
37     self.f.close()
38
39     rate = rospy.Rate(20) # 20hz
40     while not rospy.is_shutdown():
41
42         if self.is_odom == True :
43             local_path_msg=Path()
44             local_path_msg.header.frame_id='/map'
45
46             x=self.x
47             y=self.y
48             min_dis=float('inf')
49             current_waypoint=-1
50             for i,waypoint in enumerate(self.global_path_msg.poses) :
51
52                 distance=sqrt(pow(x-waypoint.pose.position.x,2)+pow(y-waypoint.pose.position.y,2))
53                 if distance < min_dis :
54                     min_dis=distance
55                     current_waypoint=i
56
57
58         if current_waypoint != -1 :
59             if current_waypoint + self.local_path_size < len(self.global_path_msg.poses):
60                 for num in range(current_waypoint,current_waypoint + self.local_path_size ) :
61                     tmp_pose=PoseStamped()
62                     tmp_pose.pose.position.x=self.global_path_msg.poses[num].pose.position.x
63                     tmp_pose.pose.position.y=self.global_path_msg.poses[num].pose.position.y
64                     tmp_pose.pose.orientation.w=1
65                     local_path_msg.poses.append(tmp_pose)
66
67             else :
68                 for num in range(current_waypoint,len(self.global_path_msg.poses) ) :
69                     tmp_pose=PoseStamped()
```

➤ 실습 2 : 경로 읽어오기 및 지역경로 생성

```
69     tmp_pose=PoseStamped()
70     tmp_pose.pose.position.x=self.global_path_msg.poses[num].pose.position.x
71     tmp_pose.pose.position.y=self.global_path_msg.poses[num].pose.position.y
72     tmp_pose.pose.orientation.w=1
73     local_path_msg.poses.append(tmp_pose)
74
75     print(x,y)
76     self.global_path_pub.publish(self.global_path_msg)
77     self.local_path_pub.publish(local_path_msg)
78     rate.sleep()
79
80
81     def odom_callback(self,msg):
82         self.is_odom=True
83         odom_quaternion=(msg.pose.pose.orientation.x,msg.pose.pose.orientation.y,msg.pose.pose.orientation.z,msg.pose.pose.orientation.w)
84         self.x=msg.pose.pose.position.x
85         self.y=msg.pose.pose.position.y
86
87
88
89     if __name__ == '__main__':
90         try:
91             test_track=path_pub_tf()
92         except rospy.ROSInterruptException:
93             pass
```

정밀도로지도

정밀도로지도

▶ 정밀도로지도란?

- 자율주행 등에 필요한 정보를 3차원으로 제작한 전자지도
 - ✓ 차선(규제선, 도로경계선, 정지선, 차로중심선)
 - ✓ 도로시설(중앙분리대, 터널, 교량, 지하차도)
 - ✓ 표지설(교통안전표지, 노면표시, 신호기)
- 해상도 0.25m 급
- 국토지리정보원에서 제작하고 배포



정밀도로지도

➤ 자동차용 지도의 종류

- 항법지도
- ADAS지도
- 정밀지도(정밀도로지도)



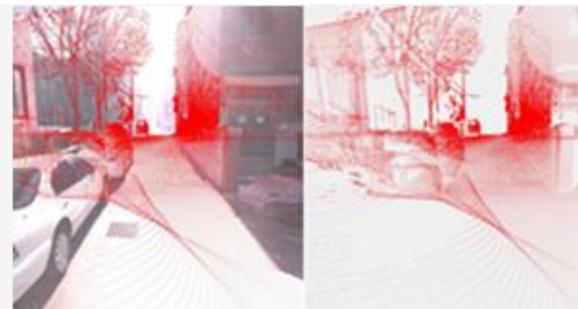
정밀도로지도

▶ 정밀도로지도 제작과정

- 다양한 센서가 결합된 MMS(Mobile Mapping System)차량을 이용
- 정밀한 위치를 측정하는 기술을 사용
 - ✓ DGPS(Differential GPS, 2개 이상의 GPS를 뜻하는 용어)
 - ✓ 관성항법장치(INS,Inertial Navigation System),
 - ✓ 차량의 이동거리를 통해 정확한 거리를 측정하는 DMI(Distance Measuring Instrument)
- 정밀한 위치에서 카메라, 라이다 등을 이용해 데이터를 캡쳐



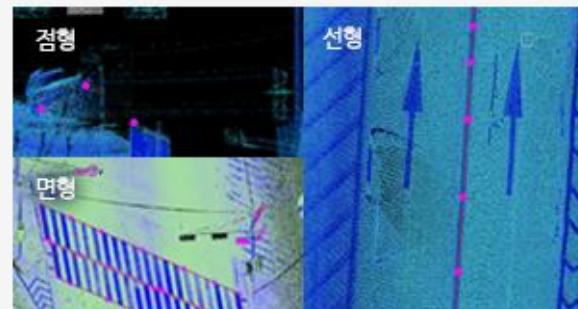
01. 작업계획



02. 데이터획득



03. 점군데이터 생성

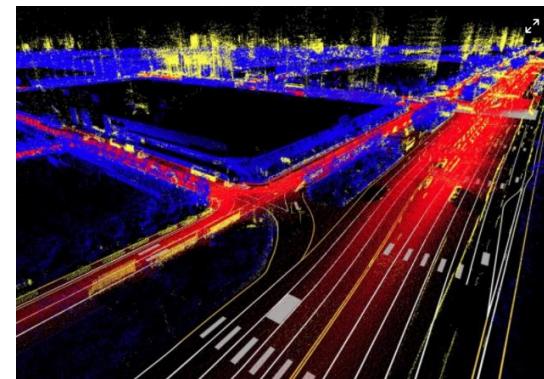
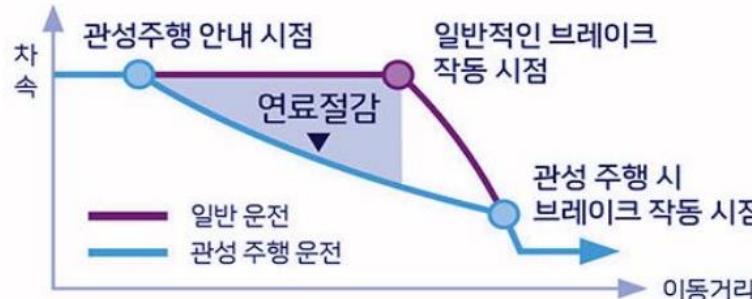


04. 객체 도화 및 편집

정밀도로지도

➤ 정밀도로지도의 필요성

- 가외성(Redundancy)을 충족시켜 신뢰성, 안전성 강화
 - ✓ 가외성은 기능, 역할의 중첩, 중복을 뜻하는 개념으로 비효율, 비능률의 의미로 통용되었다. 그러나 1969년 Martin Landua는 불확실성 하에서는 가외성이 오류발생 확률을 기하급수적으로 감소시킨다는 가외성의 타당성을 주장한 바 있다. 우리는 그 동안 가외성을 지양하고 효율성에 집중해 왔다. 그러나 자율주행 산업에서는 미세한 오차가 대형 인명사고로 이어질 수 있기 때문에 가외성이 필요하다.
- 자율주행 차량의 위치에러 감소
 - ✓ 자동차의 측위 센서와 정밀지도 정보를 매칭시키며 현재 위치 등을 보정할 수 있다.
- 친환경성 향상 및 배터리 효율관리 지원
 - ✓ 주행 도로의 경사, 곡률, 너비 등을 정밀지도를 통해 사전적으로 파악해서 최적의 운행을 할 수 있다.
- 실시간 분석해야 하는 데이터 용량 감소
 - ✓ 주행에 필요한 데이터를 미리 가지고 있기 때문에 분석해야하는 데이터가 감소된다.

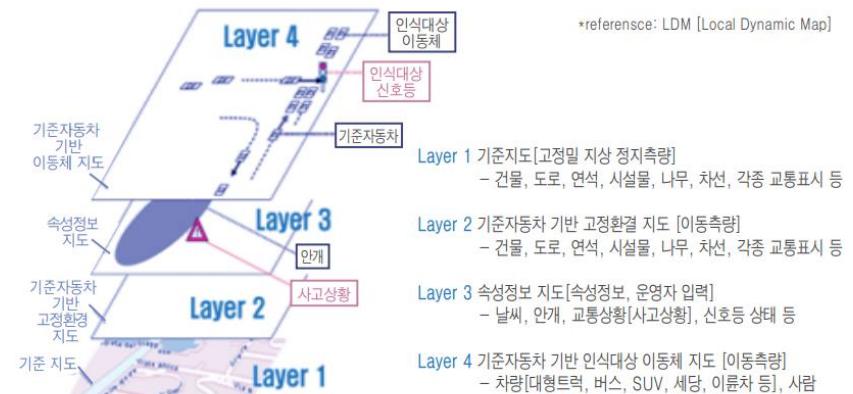
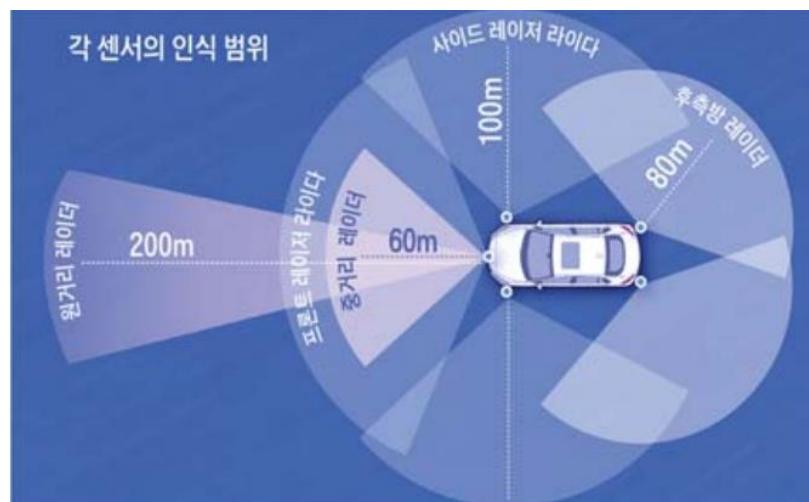


정밀도로지도

▶ 정밀도로지도의 필요성

• 인지 센서를 이용한 인지 범위의 한계 극복

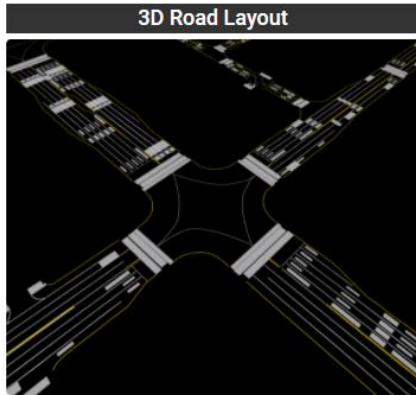
- ✓ 정확한 인지가 없다면 완전자율주행 차량은 불가능했지만, v2x와 결합해 정적인 정보 뿐만 아니라 동적 맵(Local Dynamic Map)을 이용해 인지 센서를 대체제가 아닌 보완재로 사용할 수 있다.
- ✓ 예) 시속 120km/h -> 33m/s 3~6초 안에 인지/판단/제어가 끝나야 가능하다.



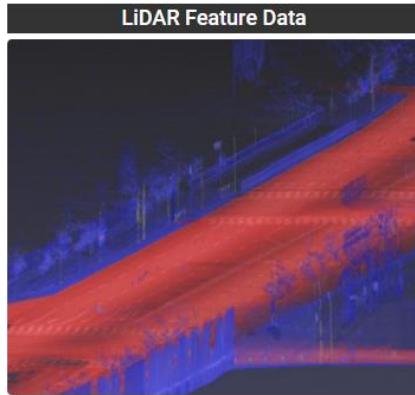
정밀도로지도

➤ 정밀도로지도 데이터

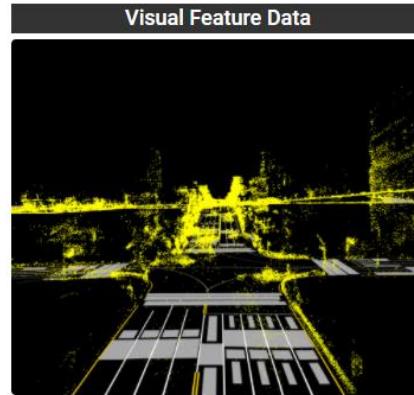
- 네이버랩스에서 공개하는 정밀지도 데이터



We provide the 3D layout of the road surface that we extract from aerial photographs. It contains information regarding the types and precise 3D locations of visual structures on the road surface that are essential for self-driving vehicles, such as lanes, road markings, crosswalks, crossroads and speed bumps.



Our HD map dataset also includes 3D LiDAR point cloud of the surrounding road environment captured from our MMS vehicle. Each point is associated with a semantic label indicating the type of object or region it belongs to. Dynamic objects like vehicles and pedestrians are automatically detected and removed from the point cloud.



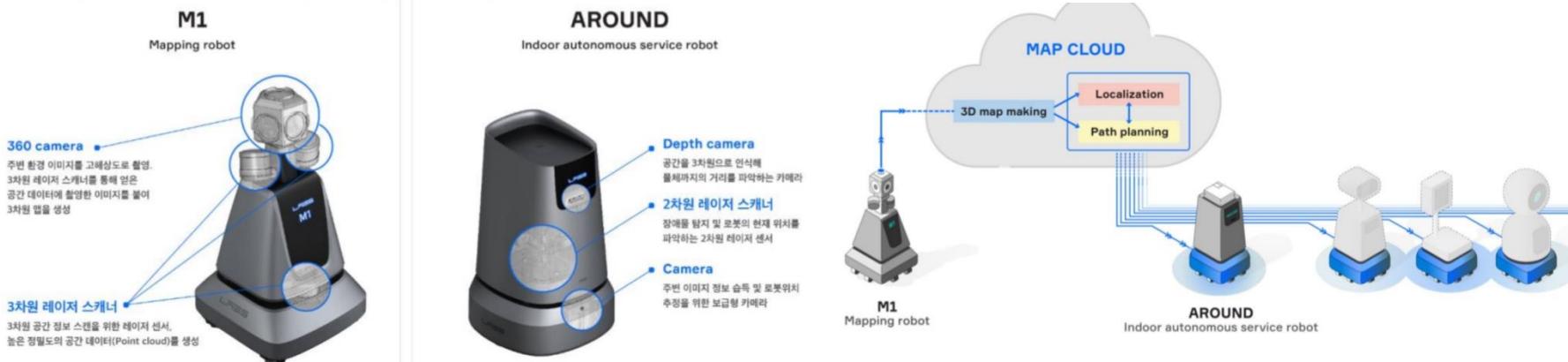
Final component of our HD map dataset is a set of visual features extracted from salient regions of the road environment. Obtained by a deep learning model trained in-house, these features are compact, discriminative, and invariant across different viewing conditions, allowing for reliable matching and localization.

Data	File Type	Dimension	Classes	QuadTile Size	Path Format
3D Road Layout	GeoJSON	[x,y,z, type]	Lane, Stop, Link, Sign, ...	-	{class}.geojson
LiDAR Feature Data	LAS	[x,y,z, intensity, classification]	Road / Pole / Other	20.48m x 20.48m x Areal UTM	{level}/{y_Idx}/{x_Idx}.las
Visual Feature Data	H5	[x,y,z, roll, pitch, yaw, feature_descriptor]	128D Feature Descriptor	20.48m x 20.48m x Areal UTM	{level}/{y_Idx}/{x_Idx}.h5

정밀도로지도

▶ 정밀지도 데이터(실내)

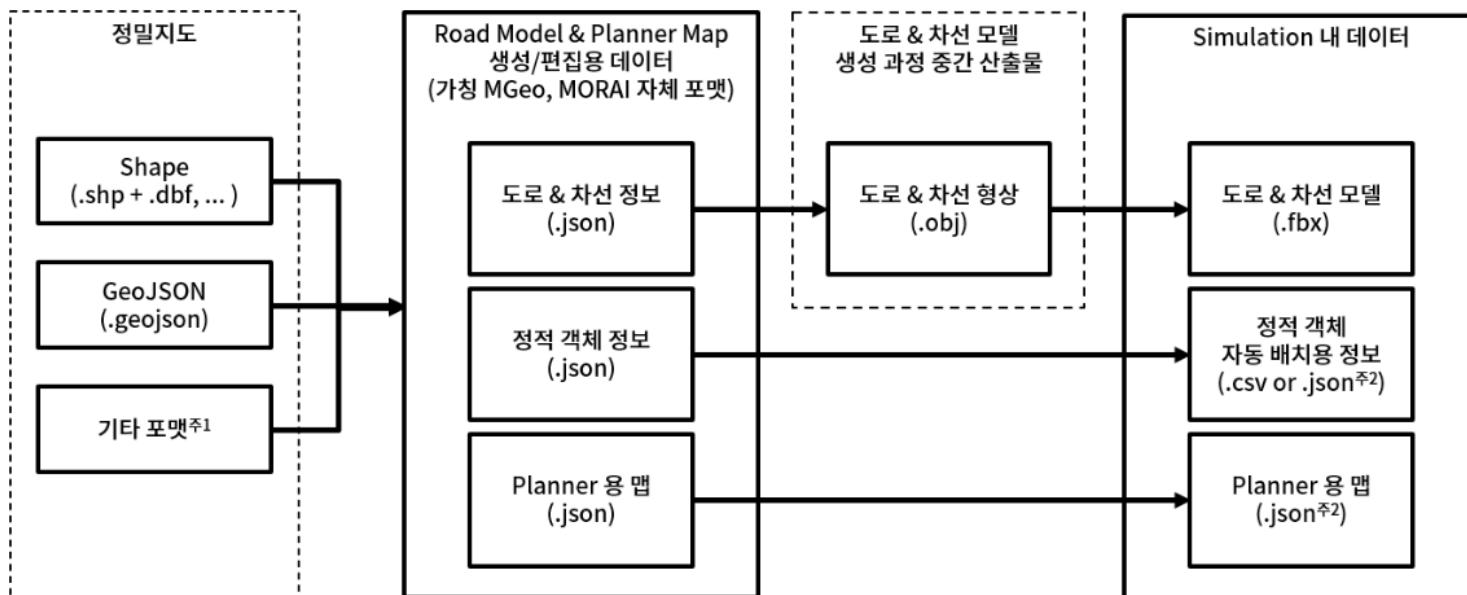
- 로봇이 실내에서 사용하는 정밀지도
- 실내 정밀지도 매핑로봇 M1
- 실내 자율주행 서비스로봇 AROUND
- 실내 AR Navigation



정밀도로지도

➤ Mgeo(MORAI Geometry)

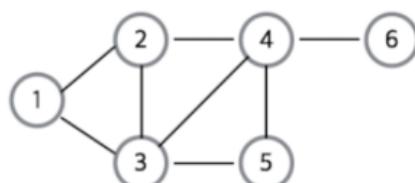
- 시뮬레이터에서 사용하는 정밀지도 포맷
- 실제와 동일한 도로 표면, 차선, 표지판, 신호등 등 생성에 사용
- 차량 주행용 경로 생성
- 정밀지도는 매우 다양한 곳에서 제작되고 그 포맷이 제각각이다. 파일 포맷도 제각각 다르며, 파일 포맷이 같더라도 실제 HDMap을 표현하는 방식 또한 다르다. 그래서 이를 보완하기 위해 만들어졌다.



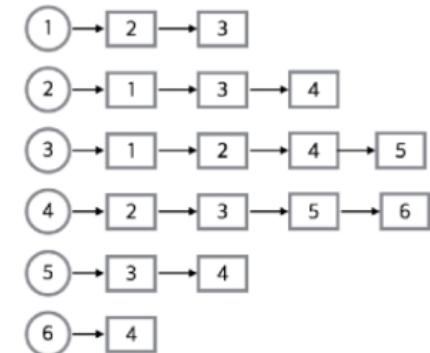
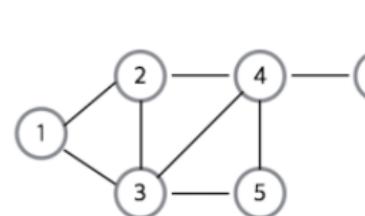
정밀도로지도

➤ Mgeo(MORAI Geometry)

- MORAI Planner 용 맵에 있는 데이터는 인접리스트 방식의 그래프 구조이다.
- 그래프란?
 - ✓ 그래프는 정점과 간선으로 이루어진 자료구조입니다. 정확히는 정점간의 관계를 표현하는 조직도라고 볼 수 있다. 대표적으로 지하철 노선도, 도심의 도로 등이 있다.
 - ✓ 정점(vertex) : 노드라고도 하며 정점에서 데이터가 저장된다.
 - ✓ 간선(edge) : 링크라고도 하며 노드간의 관계를 나타낸다.
 - ✓ 인접 정점(adjacent vertex) : 간선에 의해 연결된 정점
 - ✓ 단순 경로(simple-path) : 경로 중 반복되는 정점이 없는 것, 같은 간선을 지나가지 않는 경로
 - ✓ 차수(degree) : 무방향 그래프에서 하나의 정점에 인접한 정점의 수
 - ✓ 진출 차수(out-degree) : 방향 그래프에서 사용되는 용어로 한 노드에서 외부로 향하는 간선의 수
 - ✓ 진입 차수(in-degree) : 방향 그래프에서 사용되는 용어로 외부 노드에서 들어오는 간선의 수



	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	1	1	0	0
3	1	1	0	1	1	0
4	0	1	1	0	1	1
5	0	0	1	1	0	0
6	0	0	0	1	0	0



정밀도로지도

➤ Mgeo(MORAI Geometry)

- 노드(node)

- ✓ 필드

- ❑ idx : 노드 이름
 - ❑ to_links : 노드에서 나가는 링크 리스트
 - ❑ from_links : 노드로 들어오는 링크 리스트
 - ❑ on_stop_line : 정지선 여부 (True or False)

- ✓ 메소드

- ❑ get_to_links() : 노드에서 나가는 링크 리스트 리턴
 - ❑ get_from_links() : 노드로 들어오는 링크 리스트 리턴
 - ❑ get_to_links_idx_list() : 나가는 링크 아이디 리스트 리턴
 - ❑ get_from_links_idx_list() : 들어오는 링크 아이디 리스트 리턴
 - ❑ get_from_nodes() : 이전 연결된 노드 리스트 리턴
 - ❑ get_to_nodes() : 다음될 노드 리스트 리턴
 - ❑ is_on_stop_line() : 정지선 여부 리턴(True or False)
 - ❑ print_all_related_nodes_and_links() : 연결된 노드, 링크들 모두 출력

정밀도로지도

➤ Mgeo(MORAI Geometry)

• 링크(link)

✓ 필드

- idx : 링크이름
- from_node : 링크 시작 노드
- to_node : 링크 끝 노드
- lazy_point_init : 차선변경 링크여부
- lane_change_pair_list : 차선변경 링크 리스트
- max_speed_kph : 링크에서 주행할 수 있는 최대속도
- min_speed_kph : 링크에서 주행할 수 있는 최저속도
- traffic_signs : 신호등 연결된 신호등
- traffic_lights : 받아야하는 신호등 신호

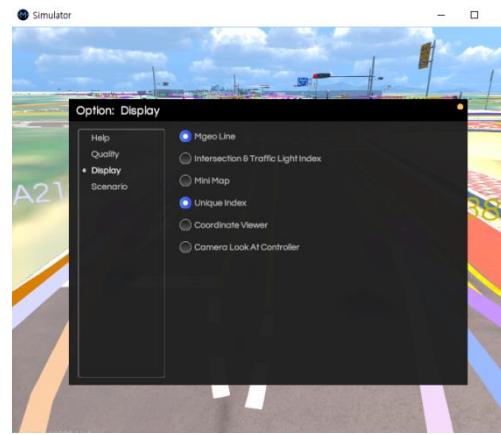
✓ 메소드

- get_to_node() : 링크 끝 노드 리턴
- get_from_node() : 링크 시작 노드 리턴
- get_to_links() : 나가는 링크 리스트 리턴
- get_from_links() : 들어오는 링크 리스트 리턴
- is_it_for_lane_change() : 차선변경 링크인지 리턴(True or False)
- get_lane_change_pair_list() : 차선변경할 수 있는 링크 리스트 리턴
- get_number_of_lane_change() : 차선변경할 수 있는 링크 갯수 리턴

정밀도로지도

➤ 시뮬레이터에서 Mgeo 확인하기

- 시뮬레이터의 Etc-> Option 메뉴를 선택하고, Display 탭에서 Mgeo Line을 클릭한다.
- 맵에 Mgeo 링크가 그려지고 링크 idx를 확인할 수 있다.



정밀도로지도

➤ Mgeo 데이터 읽어오기

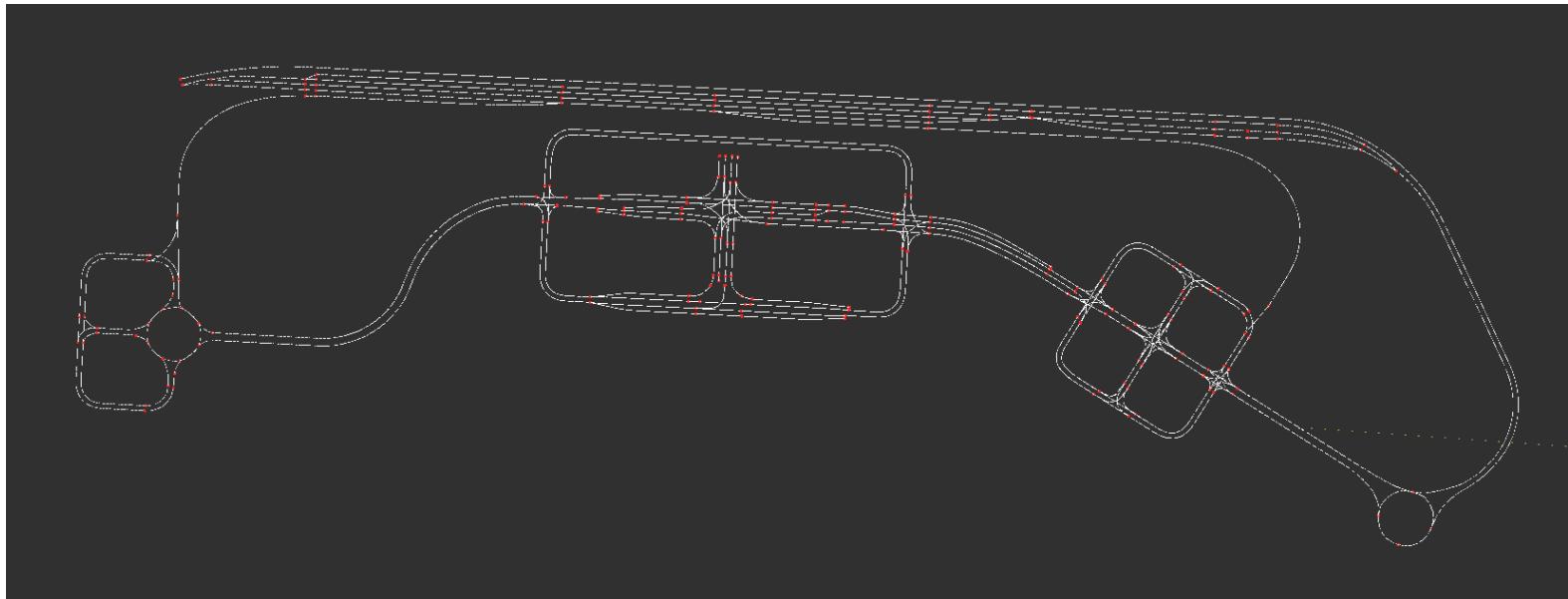
- 사용할 패키지 내에 scripts 폴더 안에 나눠준 lib 폴더를 복사
- scripts 폴더 내에 아래와 같이 파이썬 스크립트 작성 후 실행

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import os
5  import sys
6
7
8  current_path = os.path.dirname(os.path.realpath(__file__))
9  sys.path.append(current_path)
10
11 from lib.mgeo.class_defs import *
12
13
14 load_path = os.path.normpath(os.path.join(current_path, 'lib/mgeo_data/kcity'))
15 mgeo_planner_map = MGeoPlannerMap.create_instance_from_json(load_path)
16
17 node_set = mgeo_planner_map.node_set
18 link_set = mgeo_planner_map.link_set
19 nodes=node_set.nodes
20 links=link_set.lines
21 print('# of nodes: ', len(node_set.nodes))
22 print('# of links: ', len(link_set.lines))
```

정밀도로지도

➤ 실습 3 : 모든 노드, 링크 시각화하기

- Mgeo를 이용해서 노드, 링크를 RVIZ에서 시각화하기
- 각 노드, 링크들을 sensor_msgs/PointCloud 타입에 담아 Publish 하기

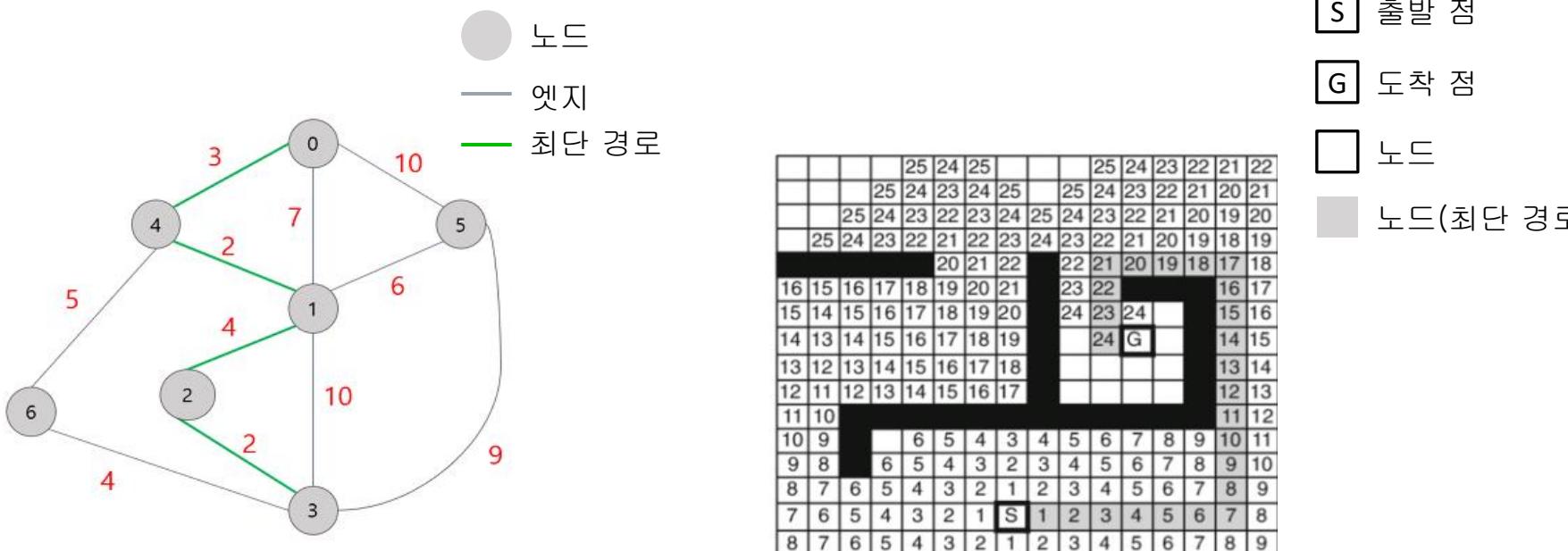


경로 탐색

경로 탐색

➤ Dijkstra 란?

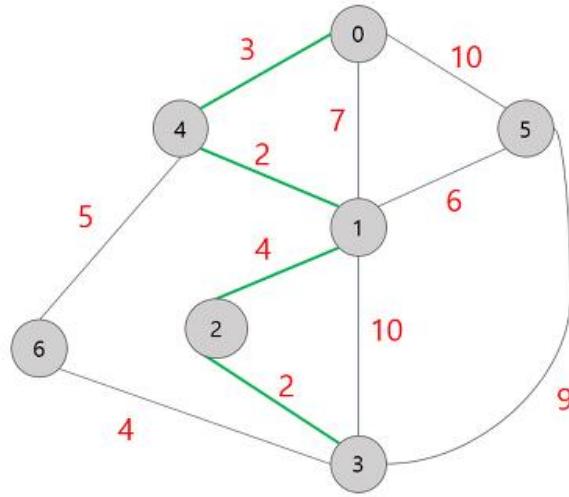
- Dijkstra는 다이나믹 프로그래밍을 활용한 대표적인 최단 경로 탐색 알고리즘이다.
 - ✓ 최단거리는 여러 개의 최단 거리로 이루어져있기 때문에 작은 문제가 큰 문제의 부분 집합에 속해있다고 볼 수 있다.
 - ✓ Dijkstra는 하나의 최단 거리를 구할 때 그 이전까지 구했던 최단 거리 정보를 그대로 사용한다.
- 특정한 하나의 정점에서 다른 모든 정점으로 가는 최단 경로를 알려주는 알고리즘이다.
- 이 때 음의 간선은 포함할 수 없다. 현실 세계에서는 음의 간선이 존재하지 않기 때문에 다익스트라는 현실 세계에서 사용하기 매우 적합한 알고리즘 중 하나이다.



경로 탐색

➤ Dijkstra 구현

- 출발지, 목적지로 가는 Dijkstra 알고리즘을 구현해보자.
- 가중치 인접 행렬을 만든다.
 - ✓ 간선이 없는 구간의 행렬값은 무한대로 놓는다.
- 최단 거리를 기록하는 1차원 배열을 만든다.
 - ✓ 이름은 distance로 하며, distance[출발지]=0 으로 놓는다.
- 방문 여부를 나타내는 1차원 배열을 만든다.
 - ✓ 이름은 s로 하며, 시작노드는 방문 체크(True)를 해준다.



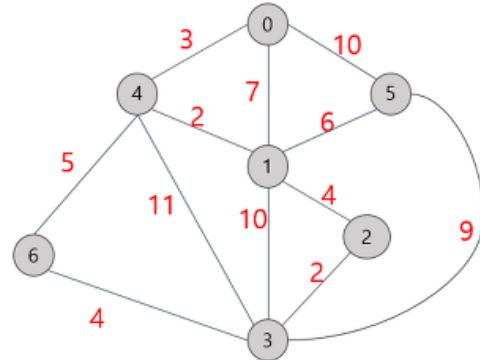
	0	1	2	3	4	5	6
0	0	7	∞	∞	3	10	∞
1	7	0	4	10	2	6	∞
2	∞	4	0	2	∞	∞	∞
3	∞	10	2	0	11	9	4
4	3	2	∞	11	0	∞	5
5	10	6	∞	9	∞	0	∞
6	∞	∞	∞	4	5	∞	0

경로 탐색

➤ Dijkstra 구현

• 노드 개수만큼 탐색

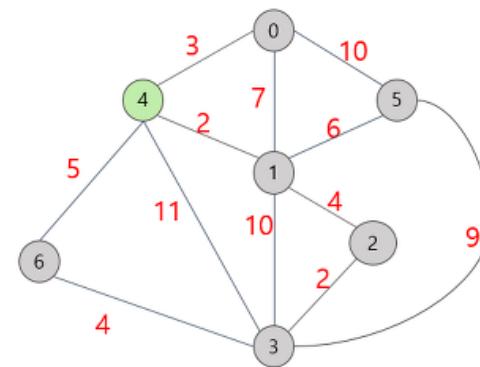
- ✓ 탐색한 적이 없으면, distance 배열에서 가장 코스트가 낮은 노드를 선택
- ✓ 선택한 노드에서 모든 노드를 탐색하면서 현재 노드까지의 거리 + 선택된 노드로 가는 간선의 길이가 distance에 저장되어있는 길이보다 작으면 업데이트



$S = \{ 0 \}$

distance[] =

0	7	∞	∞	3	10	∞
---	---	----------	----------	---	----	----------



$S = \{ 0, 4 \}$

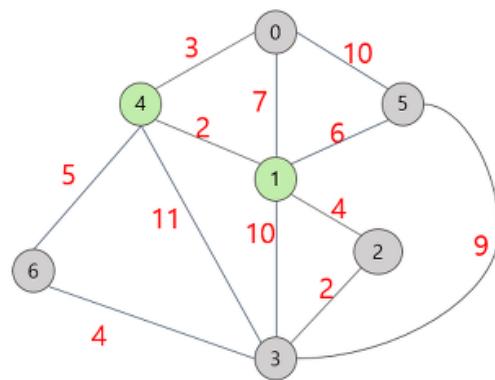
distance[] =

0	5	∞	14	3	10	8
---	---	----------	----	---	----	---

경로 탐색

➤ Dijkstra 구현

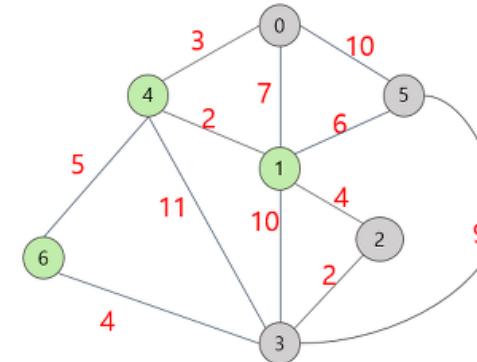
- 모든 노드를 방문할 때까지 위의 과정을 반복



$S = \{ 0, 4, 1 \}$

distance[]=

0	1	2	3	4	5	6
0	5	9	14	3	10	8



$S = \{ 0, 4, 1, 6 \}$

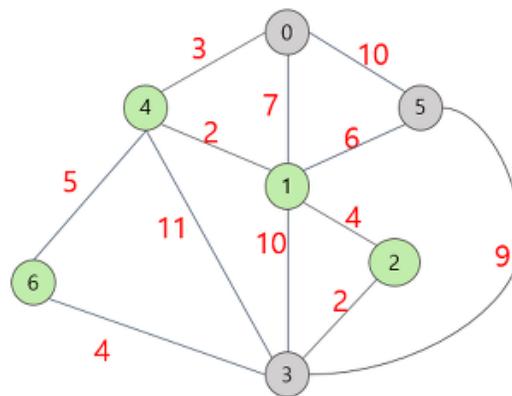
distance[]=

0	1	2	3	4	5	6
0	5	9	12	3	10	8

경로 탐색

➤ Dijkstra 구현

- 모든 노드를 방문할 때까지 위의 과정을 반복
- 3번,5번 노드는 방문해보지 않아도 유추할 수 있다
- distance 배열에 저장된 값이 출발점으로부터 각 노드로 가는 최단 경로 거리이다.



$$S = \{ 0, 4, 1, 6, 2 \}$$

distance[]=

0	5	9	11	3	10	8
---	---	---	----	---	----	---

경로 탐색

➤ Dijkstra 구현

- 최단 거리 뿐만 아니라 경로를 알아야한다.
- distance 배열을 업데이트 할 때 거쳐간 노드를 저장할 from_node 1차원배열을 생성한다.
 - ✓ from_node 배열은 초기 시작 노드로 전부 배열을 초기화 한다.
- distance 배열을 업데이트 할 때 from_node의 현재 업데이트할 노드에 현재 선택한 노드를 저장해준다.
 - ✓ from_node[탐색한 노드] = 선택한 노드
- from_node에서 목적지 노드 번호부터 역으로 출발지 노드가 나올 때까지 탐색하면 경로를 찾을 수 있다.

	0	1	2	3	4	5	6
from_node	0	4	1	2	0	0	4

경로 탐색

➤ 실습 4 : Mgeo를 이용한 최단경로 생성

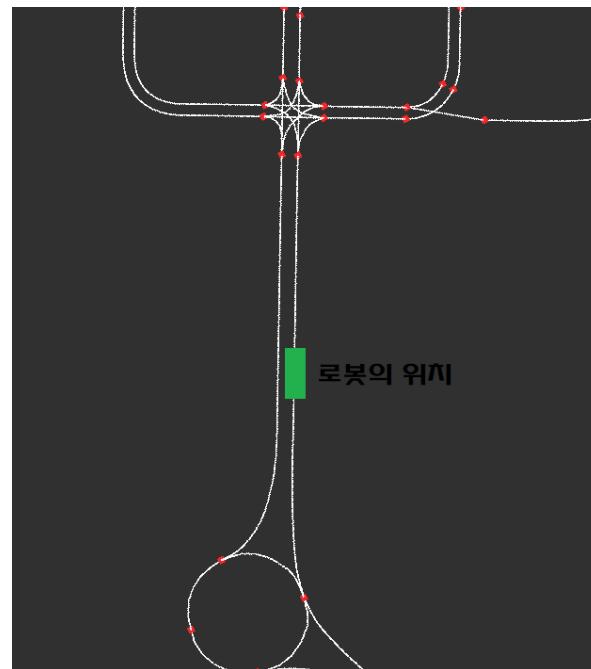
- Rviz의 2D Pose Estimate 버튼(출발지), 2D Nav Goal(목적지) 입력으로 사용
 - ✓ 출발지(토픽 : /initialpose 타입 : geometry_msgs/PoseWithCovarianceStamped)
 - ✓ 목적지(토픽 : /move_base_simple/goal 타입 : geometry_msgs/PoseWithCovarianceStamped)
 - ✓ 출발지 또는 목적지와 가장 가까운 노드를 찾아 출발지 노드, 목적지 노드를 설정
- 출발지 노드, 목적지 노드를 이용해 최단 경로 생성 후 nav_msgs/Path 타입으로 Publish



경로 탐색

➤ Advanced) 실습 4 문제점 및 해결방안

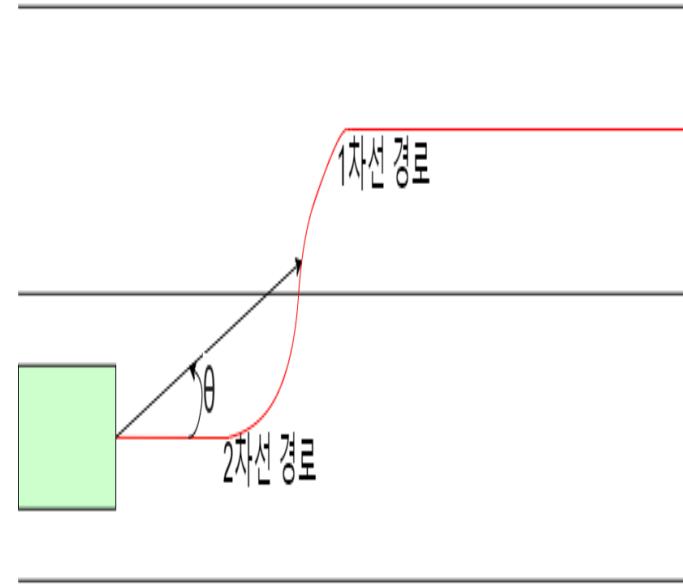
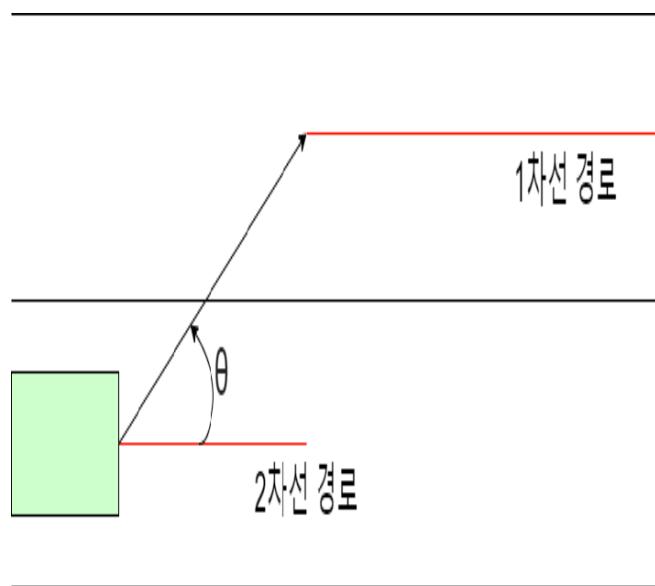
- 문제점)
 - ✓ 출발지를 직접 찍어줄 필요가 없다.
 - ✓ 거리기반으로 가까운 노드를 선택한다면 로봇이 교차로를 지나거나 애매한 위치에 존재할 경우 원하지 않는 다른 노드가 선택될 수 있다.
- 해결방안)
 - ✓ 로봇의 위치를 활용해 출발지로 사용한다.
 - ✓ 경로의 진행방향, 로봇의 헤딩을 이용해 현재 로봇이 있는 링크를 찾고, 링크의 end_node를 사용한다.



경로 탐색

▶ 차선 변경 경로 생성

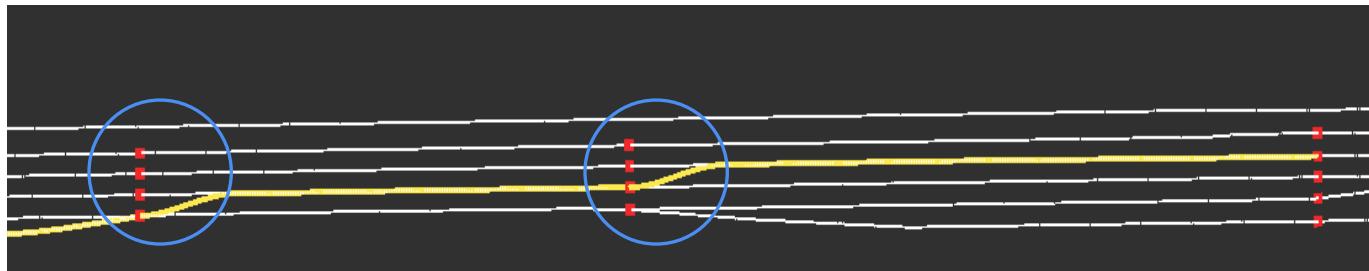
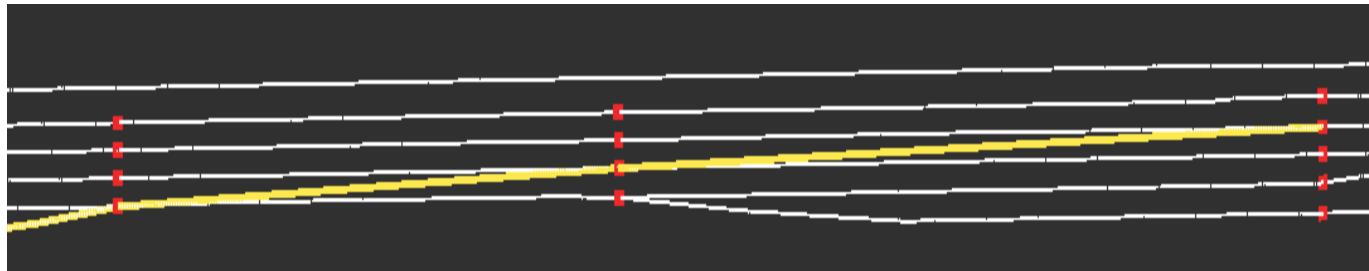
- 시뮬레이터 환경이 실제 도로와 같기 때문에 원하는 목적지로 가기 위해서는 차선변경을 해야한다.
- 경로 계획 결과를 이용해서 주행을 하게되면 차선 변경을 해야하는 구간에서 조향각이 매우 커지기 때문에 직접 차선변경 경로를 그려주어야 한다.



경로 탐색

➤ 차선 변경 경로 생성

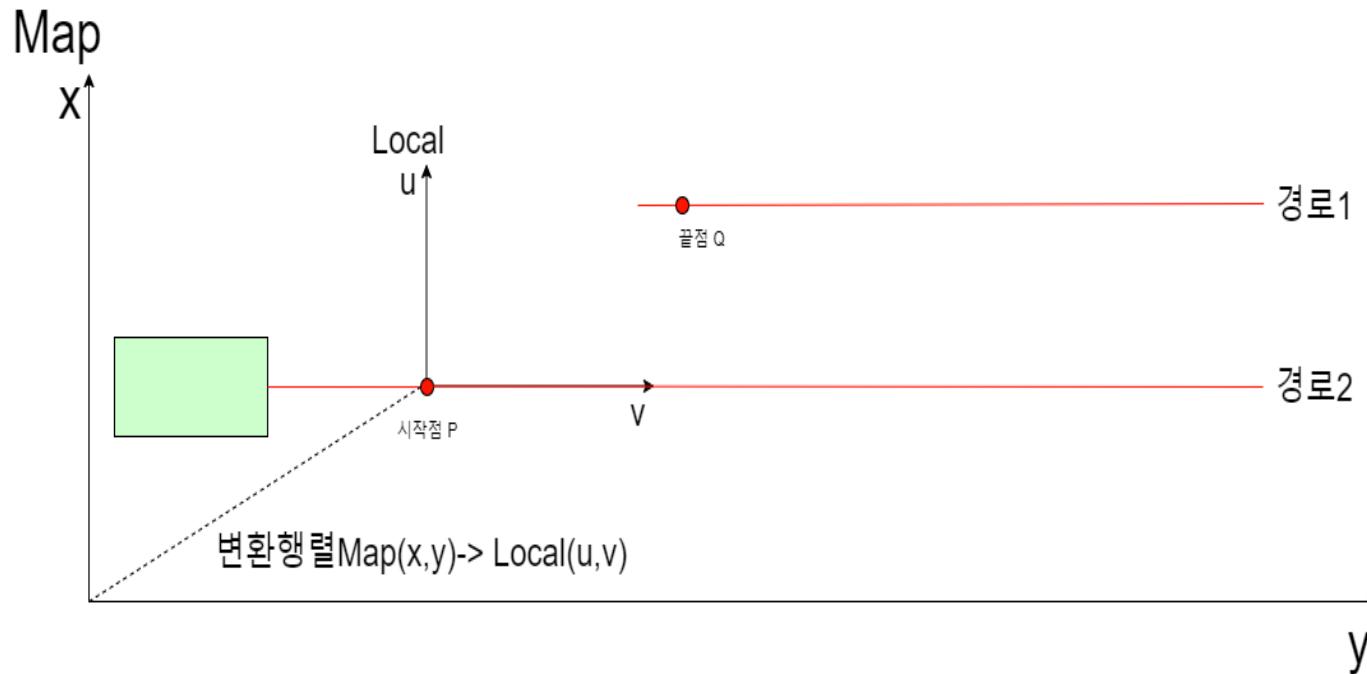
- 차선변경 경로를 직접 그려줬을 때의 결과 비교
- 차선변경 경로를 그리지 않은 링크에는 경로점들이 노드에만 찍혀있기 때문에 이어진 것처럼 보이나 경로점이 없다.



경로 탐색

▶ 차선 변경 경로 생성

- Map 좌표계의 경로에서 바꾸기 시작하는 지점(시작점), 바꾸려는 차선의 차선변경이 완료되는 지점(끝)을 Local 좌표계(u,v)로 변환 후 시작점, 끝점을 3차 곡선계획법에 의해 경로를 생성한 후 다시 Map 좌표계로 가져온다.

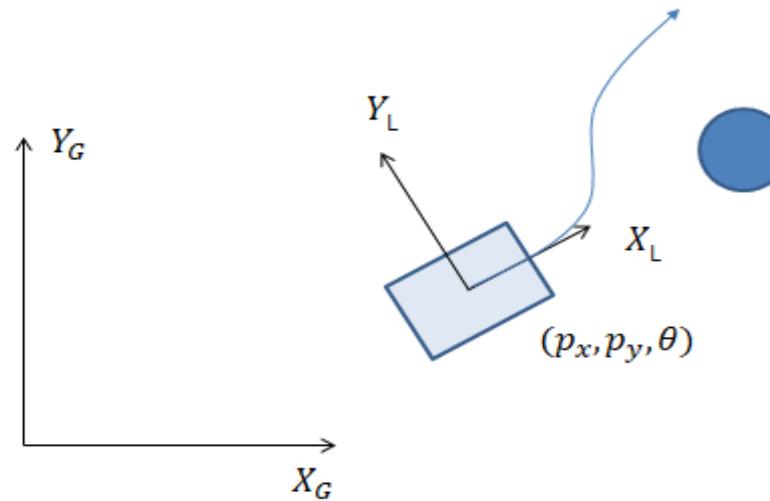


경로 탐색

➤ 차선 변경 경로 생성

- 좌표변환

- 좌표변환은 Translation & Rotation Transformation Matrix(RT Matrix)를 사용
- px,py는 경로의 위치
- theta는 경로의 진행방향



$$\begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix}_G = \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ 1 \end{bmatrix}_L$$

경로 탐색

➤ 차선 변경 경로 생성

- 좌표변환 예제코드

```
1  # -*- coding: utf-8 -*-
2  from math import cos,sin,pi,sqrt,pow,atan2
3  import numpy as np
4
5  vehicle_position_x=5
6  vehicle_position_y=5
7  heading=90.0
8
9  vehicle_yaw=heading/180*pi    # degree to radian
10
11 translation=[vehicle_position_x, vehicle_position_y]
12
13 t=np.array([
14     [cos(vehicle_yaw), -sin(vehicle_yaw),translation[0]],
15     [sin(vehicle_yaw),cos(vehicle_yaw),translation[1]],
16     [0                 ,0                 ,1           ]])
17
18 det_t=np.linalg.inv(t)
19
20 # local to global
21 local_x=5
22 local_y=0
23 local_point_vector=[local_x,local_y,1]
24 global_point_vector=t.dot(local_point_vector)
25 print(global_point_vector)
26
27
28 # global to local
29 global_x=10
30 global_y=10
31 global_point_vector=[global_x,global_y,1]
32 local_point_vector=det_t.dot(global_point_vector)
33 print(local_point_vector)
```

➤ 3차 곡선 계획법

$$U(v) = a_3v^3 + a_2v^2 + a_1v + a_0$$
$$U'(v) = 3a_3v^2 + 2a_2v + a_1$$

Initial / Final Conditions

$$U(0) = 0, U(Q_v) = Q_u$$

$$U'(0) = 0, U'(Q_v) = 0$$

$$a_0 = Q_u$$

$$a_1 = 0$$

$$a_2 = \frac{3(Q_u - P_u)}{Q_u * Q_u}$$

$$a_3 = \frac{-2(Q_u - P_u)}{Q_u * Q_u * Q_u}$$

➤ 3차 곡선 계획법

```
156 ✓    def draw_lange_change(self, start_link, end_link, lane_change_distance, step_size):
157        output_path = []
158
159        translation = [start_link.points[0][0], start_link.points[0][1]]
160        theta = atan2(start_link.points[1][1] - start_link.points[0][1], start_link.points[1][0] - start_link.points[0][0])
161
162 ✓        t = np.array([
163            [cos(theta), -sin(theta), translation[0]],
164            [sin(theta), cos(theta), translation[1]],
165            [0, 0, 1]])
166
167 ✓        det_t = np.array([
168            [t[0][0], t[1][0], -(t[0][0]*translation[0] + t[1][0]*translation[1])],
169            [t[0][1], t[1][1], -(t[0][1]*translation[0] + t[1][1]*translation[1])],
170            [0, 0, 1]])
171
172        world_end_link_list = []
173 ✓        for point in end_link.points :
174            world_end_link_list.append([point[0], point[1], 1])
175
176        world_end_link_metrix = np.array(world_end_link_list).T
177        local_end_link_metrix = det_t.dot(world_end_link_metrix).T
178
179        min_dis=float('inf')
180        local_end_point_list = []
```

➤ 3차 곡선 계획법

```
180     local_end_point_list = []
181
182     for point in local_end_link_matrix:
183         if point[0] > 0:
184             dis = abs(sqrt(point[0]*point[0] + point[1]*point[1]) - lane_change_distance)
185             if dis < min_dis:
186                 min_dis = dis
187                 local_end_point_list = [[point[0]], [point[1]] , [1]]
188
189
190     local_end_point_matrix = np.array(local_end_point_list)
191
192
193     x=[]
194     y=[]
195     x_interval=step_size
196     xs=0
197     xf=local_end_point_matrix[0][0]
198
199     ps=0.0
200     pf=local_end_point_matrix[1][0]
201
202     x_num = xf / x_interval
203     for i in range(xs, int(x_num)):
204         x.append(i * x_interval)
205
206     a = [0.0, 0.0, 0.0, 0.0]
207     a[0] = ps
208     a[1] = 0
209     a[2] = 3.0 * (pf - ps) / (xf**2)
210     a[3] = -2.0 * (pf - ps) / (xf**3)
```

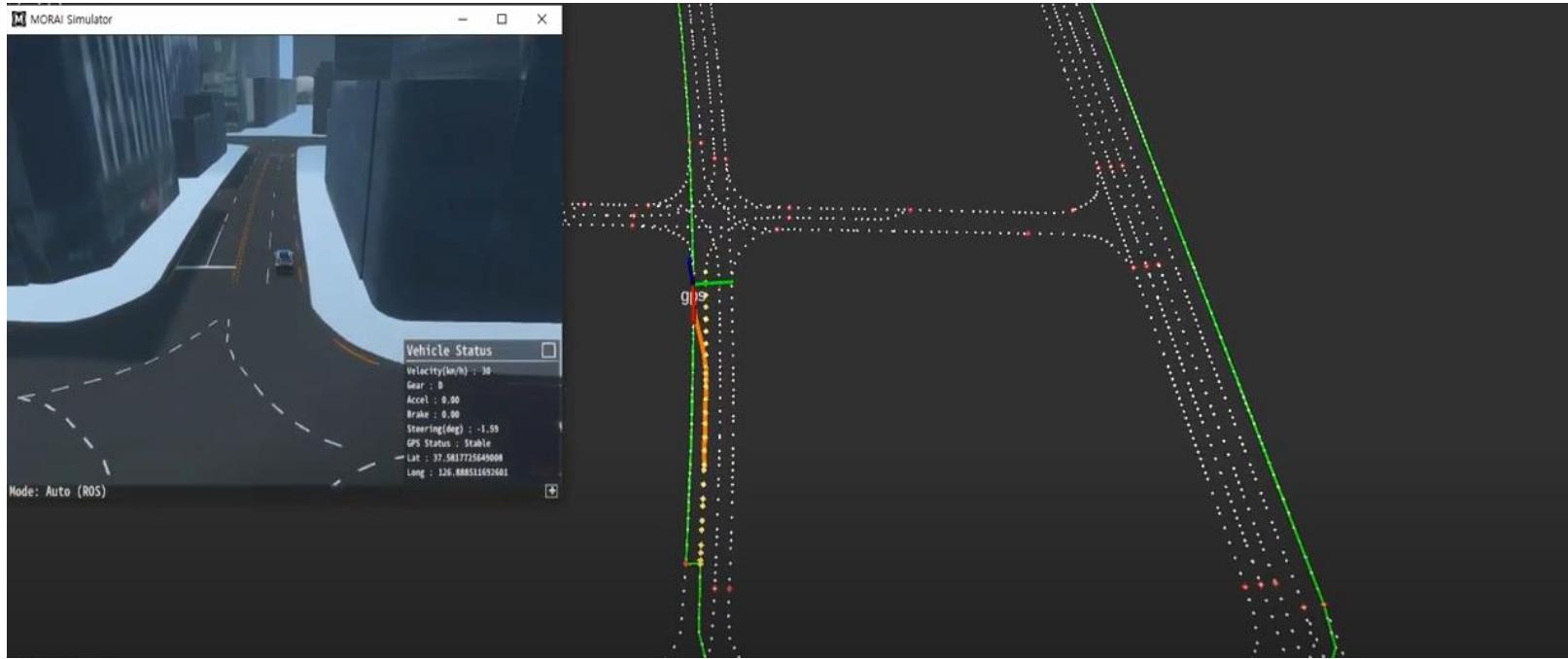
➤ 3차 곡선 계획법

```
210     a[3] = -2.0 * (pf - ps) / (xf**3)
211     for i in x:
212         result=a[3]*i**3 + a[2]*i**2 + a[1]*i + a[0]
213         y.append(result)
214
215     for i in range(0, len(y)) :
216         local_change_path = np.array([[x[i]],[y[i]],[1]])
217         global_change_path = t.dot(local_change_path)
218         output_path.append([global_change_path[0][0], global_change_path[1][0],0])
219
220
221     end_point_index = 0
222     for (i,end_point) in enumerate(local_end_link_metrix.tolist()):
223         if end_point[0] == local_end_point_matrix[0][0] and end_point[1] == local_end_point_matrix[1][0] :
224             end_point_index = i
225             break
226
227
228     for end_point in end_link.points[end_point_index:]:
229         output_path.append([end_point[0],end_point[1],0])
230
231     return output_path
232
```

경로 탐색

➤ 차선변경 판단

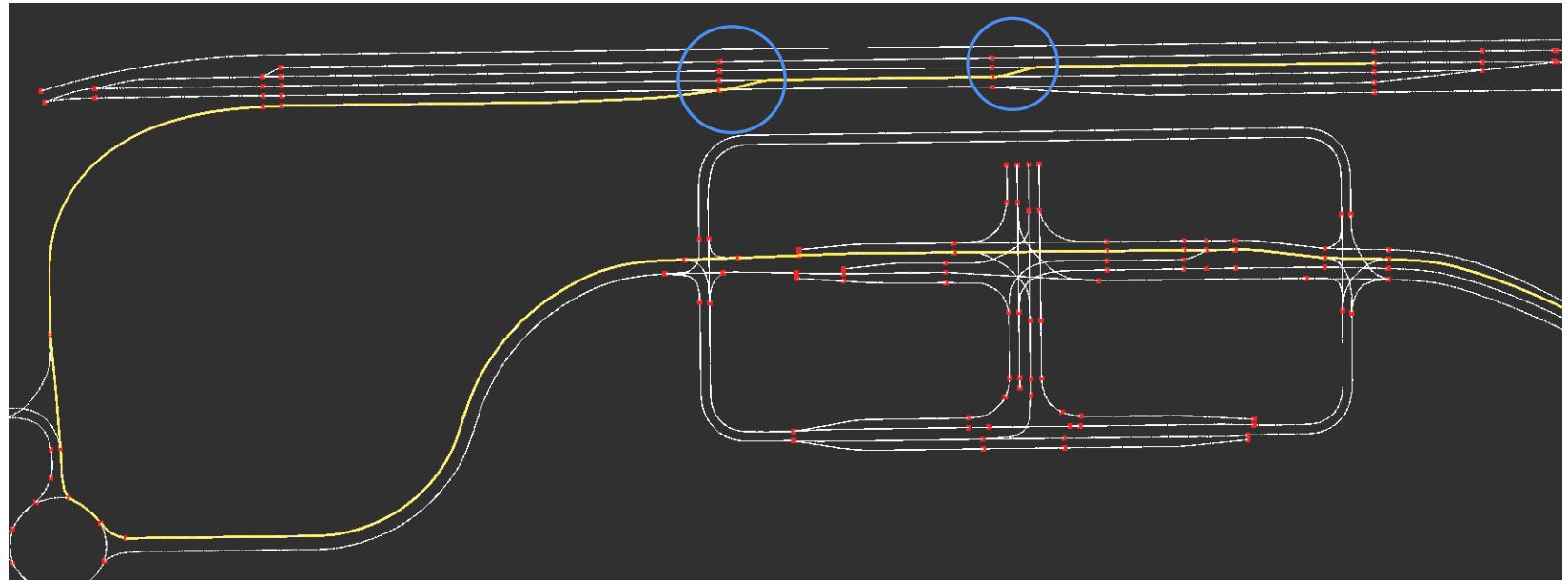
- 차선변경 판단 여부까지 고려해서 차선변경 경로 생성하기



https://www.youtube.com/watch?v=0vifRy_Zc1w

경로 탐색

- 실습 5 : 차선변경을 고려한 전역 경로 생성
 - 기존 최단 경로 생성 알고리즘(실습4)에서 차선변경할 링크와 차선변경전 링크를 찾아서 차선변경까지 고려한 전역경로 생성하기



경로 탐색

➤ Advanced) 실습 5 문제점 및 해결방안

- 차선변경 구간을 이미 전역경로에서 정해서 사용한다는 것은 항상 같은 구간에서 차선변경을 해야하는 것을 의미함
- 실제 도로에서는 항상 같은 구간에서 차선 변경하는 것이 불가능함
- 따라서 차선 변경할 수 있는지 판단을 한 후에 차선변경 경로를 그려주는게 맞다.

제어

제어

▶ 제어란?

- 대상 물체를 원하는 대로 조작 하는 것



☞ 리모컨을 이용하여 TV를 원하는 대로 조작(제어)할 수 있다.



☞ 게임 컨트롤러를 이용하여 게임 속 캐릭터를 조작(제어) 할 수 있다.

제어

➤ 로봇의 제어란?

- 로봇을 원하는 위치, 상태(속도, 가속도 등)로 조작 하는 것



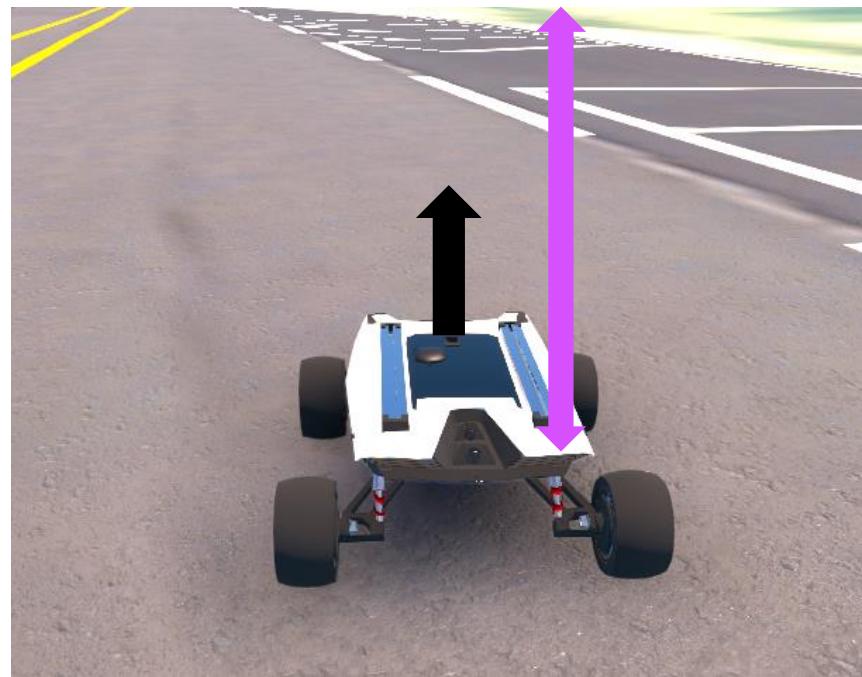
제어

➤ 종방향 제어

- 물체가 바라보는 방향 기준으로 수평한 방향의 제어를 말한다.
 - ✓ 스카우트에서는 앞/뒤로 움직이는 제어를 종 방향 제어라고 한다.
- 횡방향 제어를 통해 경로를 추종한다.

↔ 종 방향 제어

← 터틀봇이 바라보는 방향



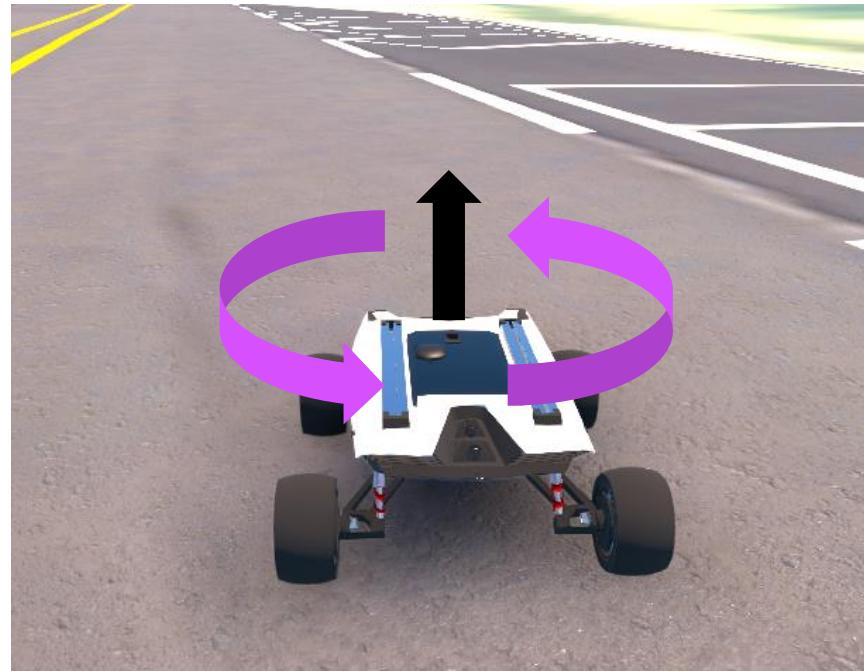
제어

➤ 횡방향 제어

- 물체가 바라보는 방향 기준으로 수직한 방향의 제어를 말한다.
 - ✓ 스카우트에서는 좌/우로 움직이는 제어를 횡 방향 제어라고 한다.
- 횡방향 제어를 통해 경로를 추종한다.

↔ 종 방향 제어

← 터틀봇이 바라보는 방향



➤ 스카우트 ROS 통신

- 스카우트 제어 메시지
 - ✓ 토픽 : /cmd_vel
 - ✓ 타입 : geometry_msgs/Twist
 - ✓ linear.x = 선속도(m/s)
 - ✓ angular.z = 각속도(rad/s)

- 스카우트 상태 메시지
 - ✓ 토픽 : /scout_status
 - ✓ 타입 : scout_msgs/ScoutStatus
 - ✓ linear_velocity = 선속도(m/s)
 - ✓ angular_velocity = 각속도(rad/s)

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import rospy
4  from scout_msgs.msg import ScoutStatus
5  from geometry_msgs.msg import Twist
6
7  class scout_controller :
8      def __init__(self):
9          rospy.init_node('scout_controller', anonymous=True)
10         self.cmd_vel_pub = rospy.Publisher('/cmd_vel',Twist, queue_size=1)
11         rospy.Subscriber('/scout_status', ScoutStatus, self.status_callback)
12         scout_cmd_vel_msg=Twist()
13
14
15     while not rospy.is_shutdown():
16         # # 직진
17         scout_cmd_vel_msg.angular.z=0.0
18         scout_cmd_vel_msg.linear.x=1.0
19
20         # # 좌회전
21         # scout_cmd_vel_msg.angular.z=0.3
22         # scout_cmd_vel_msg.linear.x=-1.0
23
24         # # 제자리 회전
25         # scout_cmd_vel_msg.angular.z=1.0
26         # scout_cmd_vel_msg.linear.x=0.0
27         self.cmd_vel_pub.publish(scout_cmd_vel_msg)
28         rate.sleep()
29
30     def status_callback(self,msg):
31         #선속도, 각속도
32         print(msg.linear_velocity, msg.angular_velocity)
33
34 if __name__ == '__main__':
35     test=scout_controller()

```

제어

➤ 조향 장치에 따른 횡 방향 제어 방법



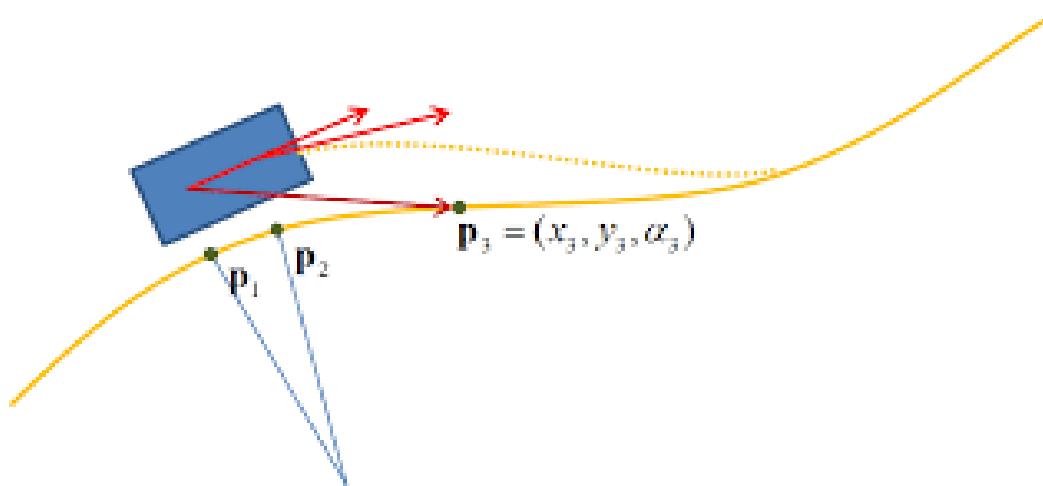
☞ 조향 장치가 없는 경우



☞ 조향 장치가 있는 경우

▶ 경로추종

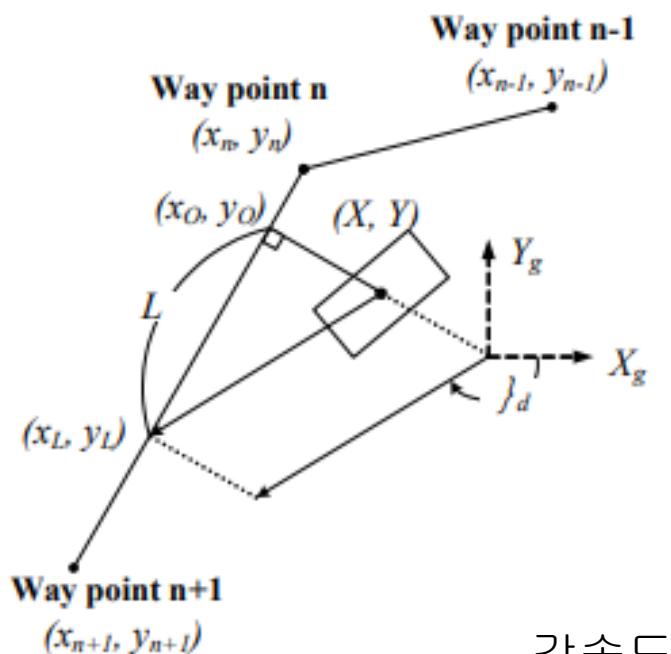
- 경로 추종은 주어진 경로와 로봇간에 위치오차와 방향오차를 줄이도록 제어하는 것이 목표다.
- 위치오차는 기준경로에서 이동로봇이 떨어진 정도이며, 방향오차는 기준경로의 진행방향과 로봇간의 방향각간의 편차이다.
- 크게 분류하자면 기하학적 방법, 모델링적 방법 두 가지로 나눌 수 있다.



제어

▶ 경로추종

- Follow the carrot
 - ✓ 현재 위치와 목표지점까지의 오차 각도와 거리(직선)이용
 - ✓ 전방주시거리만 가지고 조향각을 간단하게 계산 가능



$$u_p = \frac{(X - x_n)(x_{n+1} - x_n) + (Y - y_n)(y_{n+1} - y_n)}{(x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2}$$

$$\begin{aligned}x_O &= x_n + u_p(x_{n+1} - x_n) \\y_O &= y_n + u_p(y_{n+1} - y_n)\end{aligned}$$

$$\begin{aligned}(x_L, y_L) &= (x_O, y_O) + L \\ \phi_d &= \text{atan2}(y_L - Y, x_L - X)\end{aligned}$$

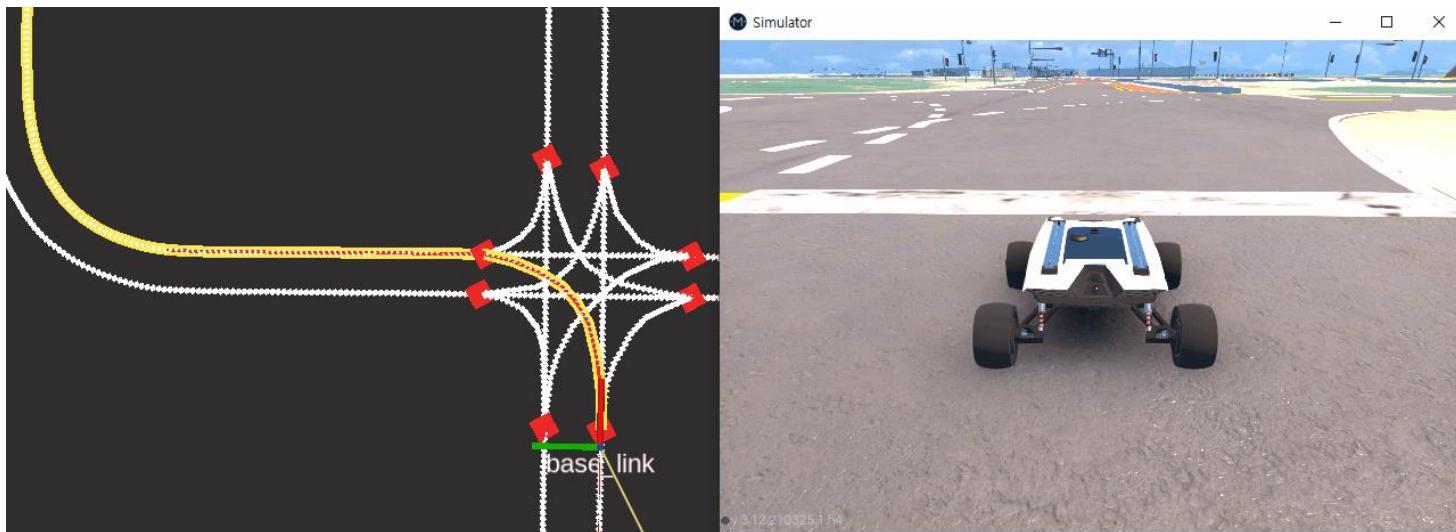
$$\omega = \frac{\phi_d}{T}$$

← 제어주기

☞ Follow the Carrot 수식

➤ 실습 6 : 경로 추종

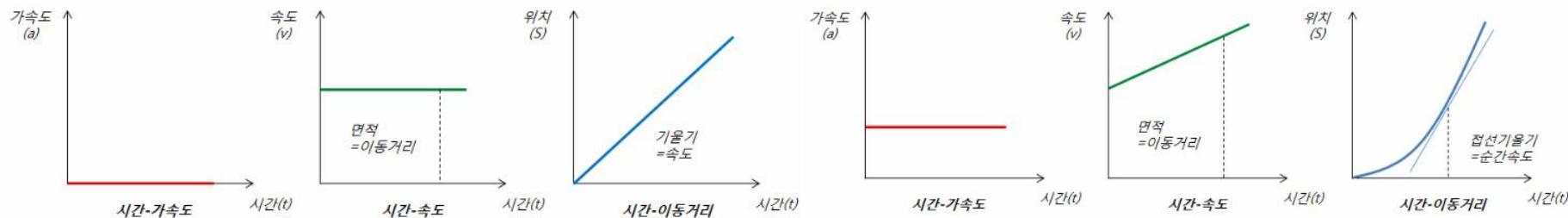
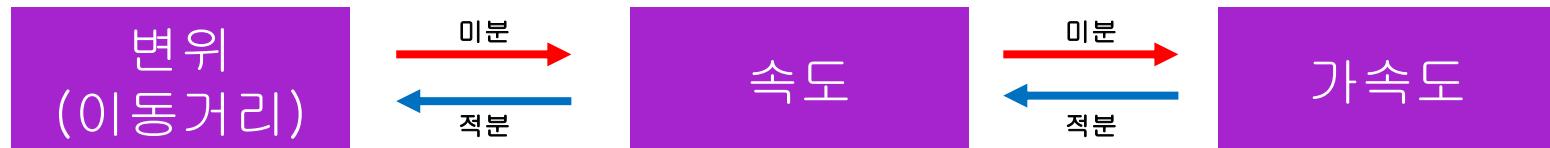
- Follow the carrot을 이용한 경로추종 알고리즘 구현
- 종방향 제어는 하지 않는다.(선속도는 상수)
- 경로를 잘 추종하는 전방주시거리 설정



제어

▶ 종방향 제어

- 종방향 제어를 위한 기본 개념인 이동거리, 속도, 가속도의 관계
 - ✓ 이동거리 - 속도 - 가속도 3가지는 이동체의 움직임을 표현하거나 분석하는데 있어 가장 기본적인 단위들
 - ✓ 위 3가지 중 최소 1가지만이라도 알 수 있는 경우 미분/적분을 통해 나머지 2가지를 연산 및 추정할 수 있음



☞ 등속도 운동할 때의 거리-속도-가속도 그래프

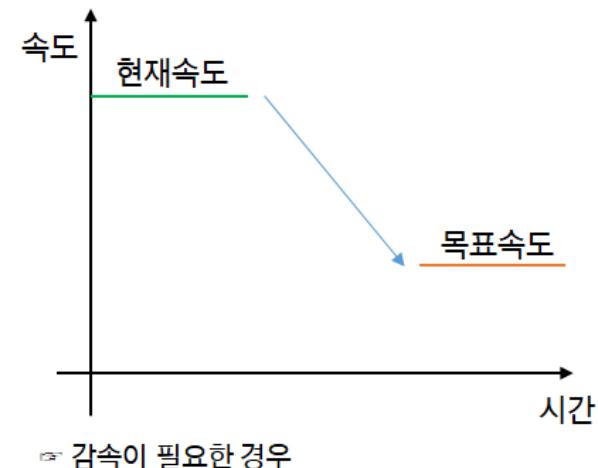
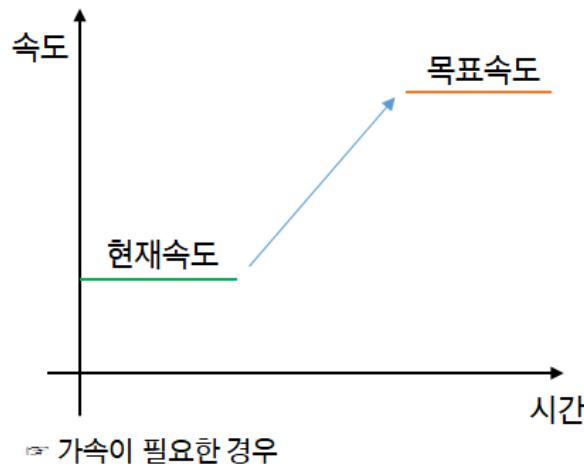
☞ 등가속도 운동할 때의 거리-속도-가속도 그래프

제어

▶ 종방향 제어

• 피드백 제어

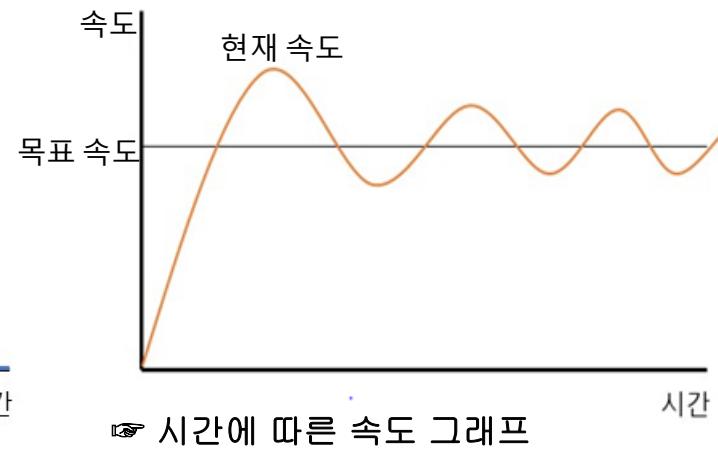
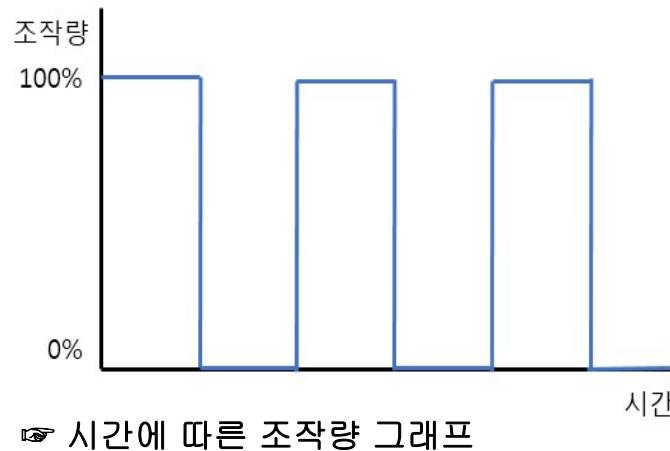
- ✓ 현재 속도와 목표속도 차이에 따라 가속 또는 감속을 한다.
- ✓ 피드백 제어를 통해 목표속도에 도달할 수 있다.
 - ON/OFF 제어
 - PID 제어



제어

➤ 종방향 제어

- **ON/OFF 제어**
 - ✓ 단순히 목표속도에 도달하기 위해 조작을 ON/OFF 하는 제어
 - ✓ 목표속도로 완벽하게 수렴하지 못한다.

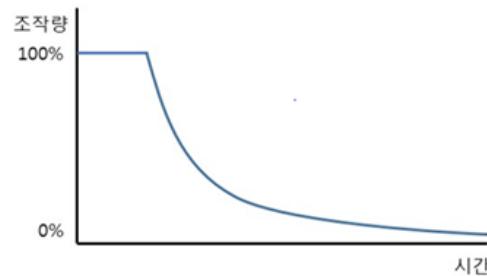
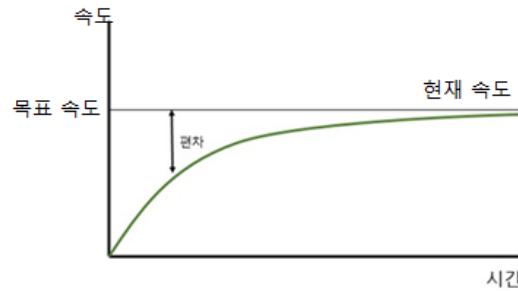


제어

▶ 종방향 제어

• P (Proportional) 제어

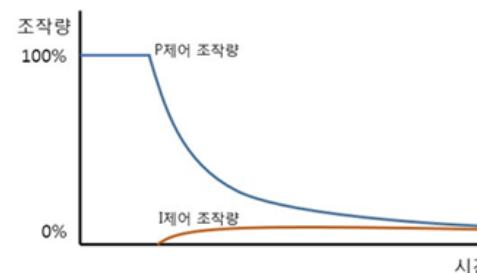
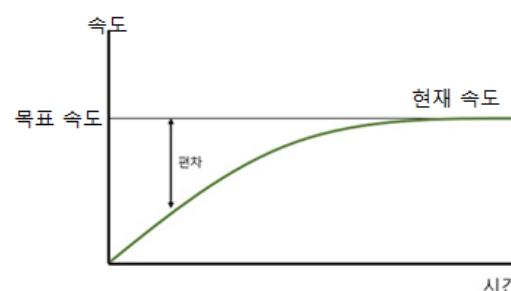
- 오차는 목표속도-현재속도이다.
- 오차에 비례하는 조작량을 가한다.
- 오차를 완전히 줄일 수 없다.



☞ P 제어 속도, 조작량 그래프

• PI(Proportional-Integral) 제어

- 비례제어, 적분 제어이다.
- P제어 조작량+ 오차를 누적한 조작량을 더해서 제어한다.
- 오차를 완전히 줄일 수 있다.



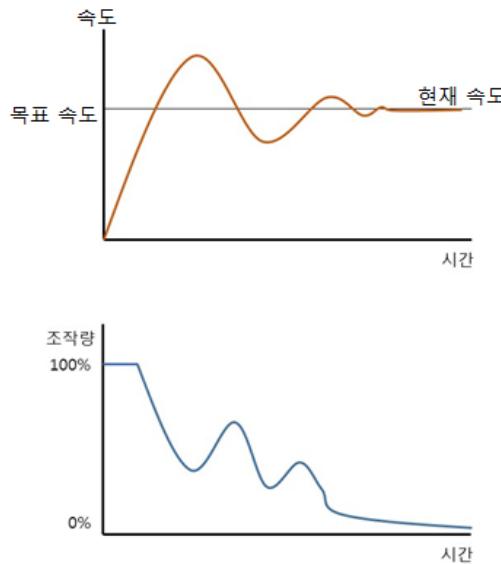
☞ PI 제어 속도, 조작량 그래프

제어

➤ 종방향 제어

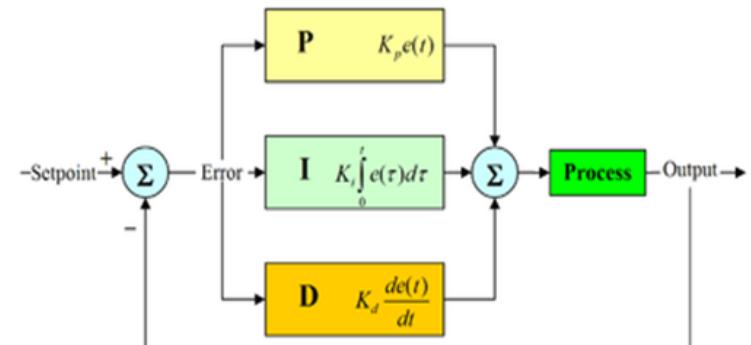
• PID(Proportional-Integral-Defferential) 제어

- 원하는 값에 도달하기 위한 기초적인 피드백 제어 방법 중 하나이다.
- P, PI, PD, PID 등 제어 대상에 맞게 선택해서 사용할 수 있다.
- PID 제어는 수식이 매우 간단하고, 구현 난이도 대비 목표치 추종이나 외란 감쇄 효과에 탁월한 성능을 가진다.
- 적절한 이득값(K_P, K_I, K_D) 조절이 필요하다.



☞ PID 제어 속도, 조작량 그래프

$$u(t) = K_P e(t) + K_I \int e(t) dt + K_D \frac{de(t)}{dt}$$



☞ PID 제어 블록선도 및 수식

➤ 종방향 제어

- **PID 이득값 튜닝**
 - ✓ **비례항** : 현재 상태에서의 오차 값의 크기에 비례한 제어작용을 한다.
 - ✓ **적분항** : 정상상태 오차를 없애는 작용을 한다.
 - ✓ **미분항** : 출력값의 급격한 변화에 제동을 걸어 오버슛을 줄이고 안정성을 향상시킨다.
 - ✓ **PID 성능은 주로 아래 4가지의 지표를 가지고 측정한다.**
 - **Rise Time** : 목표 값의 10%에서 90%까지 도달하는데 걸리는 시간
 - **Overshoot** : 현재 값이 목표 값보다 커졌을 때의 값
 - **Setting Time** : 목표 값의 5%이내에 들어갈 때의 시간
 - **Steady-State Error** : 정상상태에 도달하고 나서 존재하는 에러

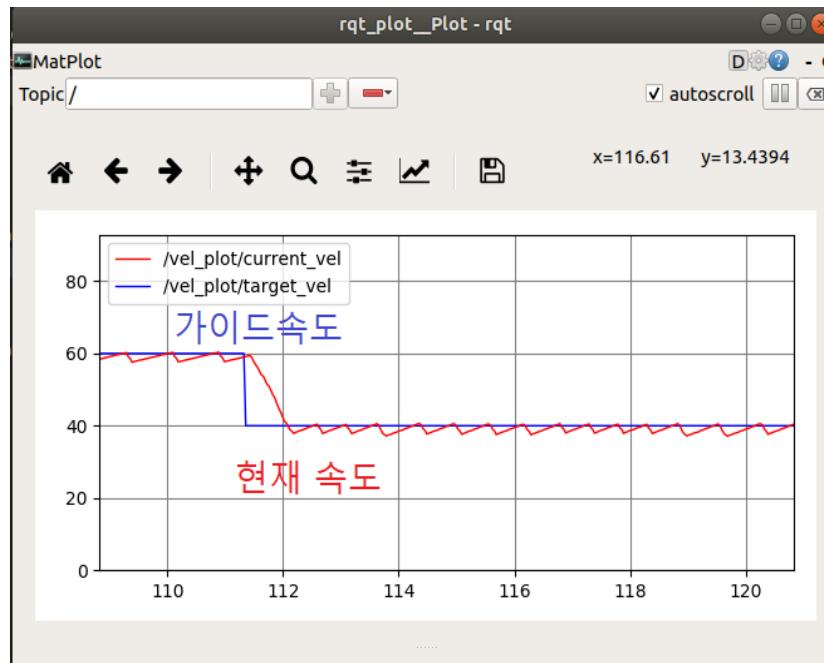
CL Response	Rise Time	Overshoot	Settling Time	Steady-State Error
KP	Decrease	Increase	Small change	Decrease
KI	Decrease	Increase	Increase	Eliminate
KD	Small change	Decrease	Decrease	No change

☞ 제어 이득값 튜닝에 따른 반응 예시

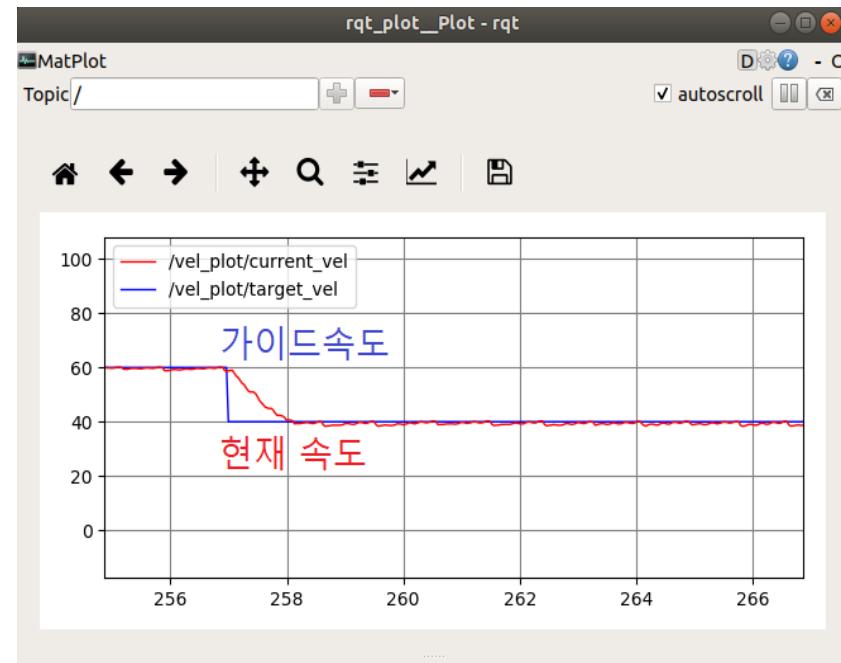
제어

➤ 종방향 제어

- PID 제어 결과 예시
 - ✓ PID 적용 결과 가이드 속도(목표 속도)에 더 빠르게 도달하고, 에러가 적은 것을 볼 수 있다.



☞ ON/OFF 제어 결과

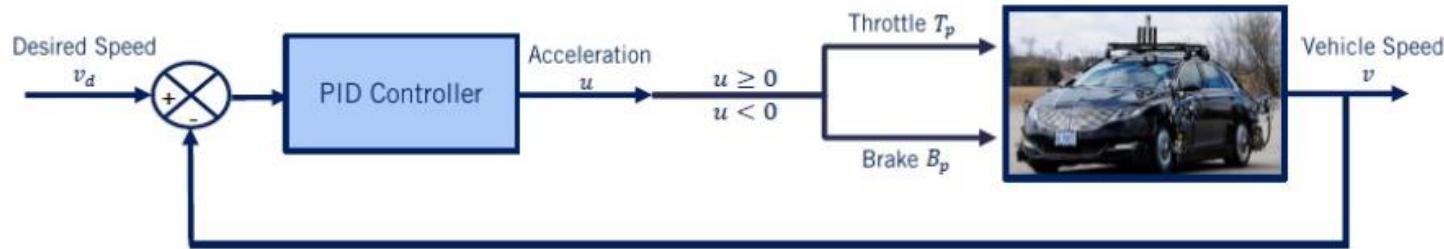


☞ PID 제어 결과

➤ 종방향 제어

- 실제 차량에 적용 예시

Longitudinal Control



- Desired speed (v_d) Vehicle speed (v) Acceleration input (u)
- No low level controller details required

$$u = K_P(v_d - v) + K_I \int_0^t (v_d - v) dt + K_D \frac{d(v_d - v)}{dt}$$

- Throttle position (T_p) Brake position (B_p)
- If $u \geq 0$: $T_p = u$, $B_p = 0$
- If $u < 0$: $T_p = 0$, $B_p = -u$

➤ 실습 7 : 종방향 제어기 설계

- 목표속도로 빠르게 도달할 수 있는 종방향 제어기 설계
- 적절한 PID게인을 사용해 Scout 미니가 목표한 속도로 빠르게 수렴하도록 한다.
- Scout의 상태 메시지, 제어 메시지를 사용
- 차량의 목표속도는 커맨드라인을 통해 직접 메시지를 publish 해준다.
 - ✓ ex) rostopic pub /target_velocity std_msgs/Float32 "data: 5.0"

속도 계획

속도 계획

➤ 속도 계획이란?

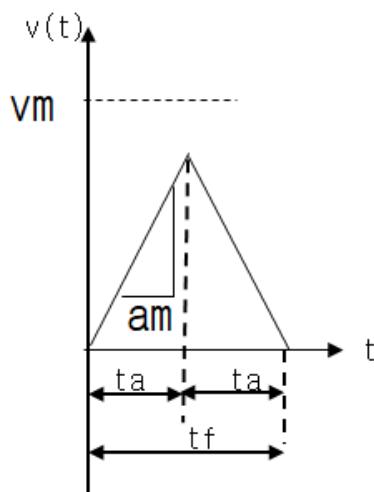
- 로봇이 어떤 속도로 주행할 것 인지에 대한 계획
- 내부적 요인(차량의 최대 가속도, 최대 감속도, 최대 속도), 외부적 요인(도로 곡류, 노면상태, 제한속도 구역), 미션을 모두 고려
- 사다리꼴 속도 계획법(내부적 요인을 이용)
- 도로 곡률기반 속도 계획법(외부적 요인을 이용)
- 미션(AEB, ACC)
- 속도 계획 결과 가장 낮은 속도를 선택해서 사용하는게 안전하다.

속도 계획

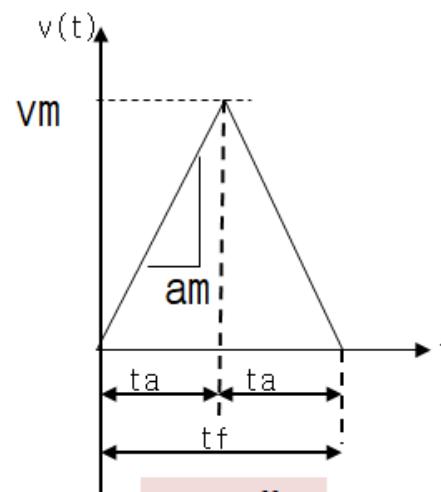
▶ 사다리꼴 속도 계획법

- 차량의 내부 요인과 이동할 거리를 가지고 속도 계획

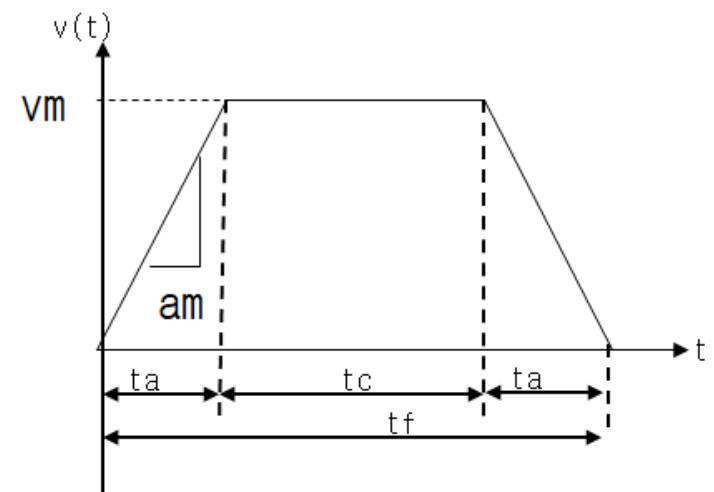
$$|p_f - p_i| < \frac{v_m^2}{a_m} \text{ 일 때}$$



$$|p_f - p_i| = \frac{v_m^2}{a_m} \text{ 일 때}$$



$$|p_f - p_i| > \frac{v_m^2}{a_m} \text{ 일 때}$$



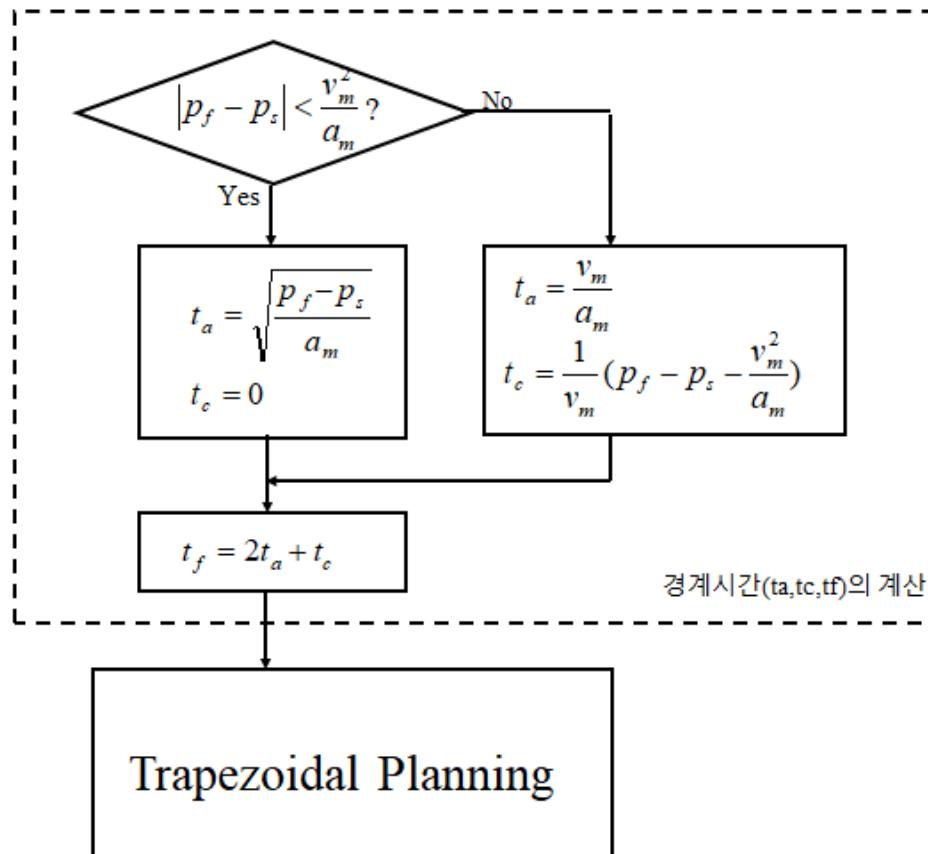
Case1: 등속구간 없음($t_c=0$)

Case2: 등속구간 있음($t_c>0$)

속도 계획

▶ 사다리꼴 속도 계획법

- 경계 시간을 계산한다.

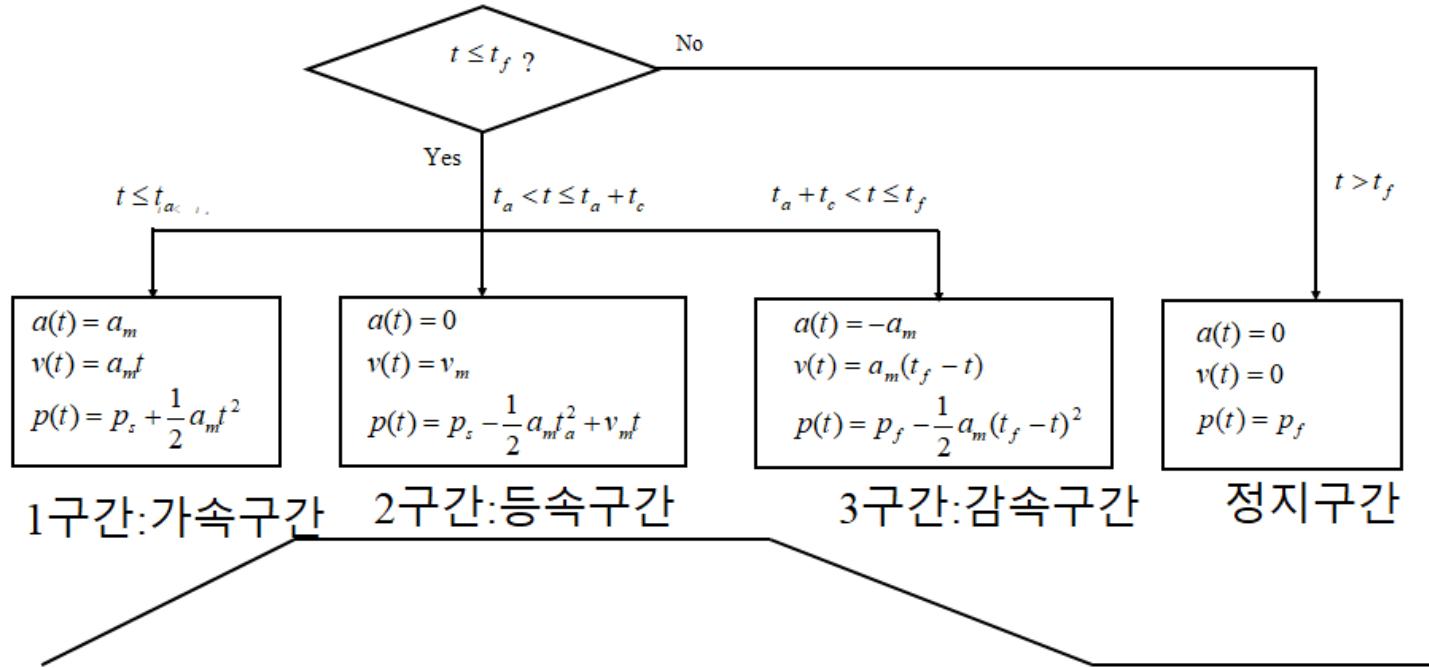


속도 계획

▶ 사다리꼴 속도 계획법

- 구간별 속도, 위치 계산

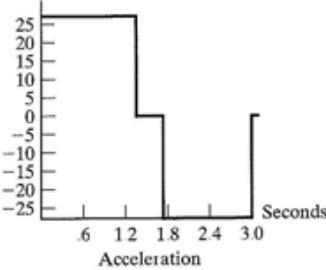
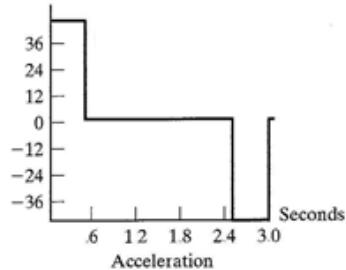
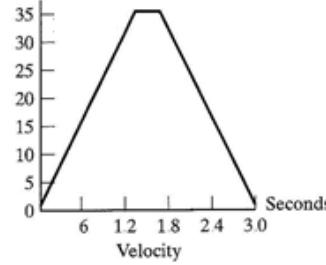
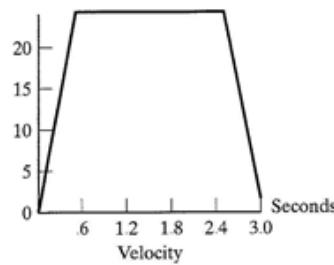
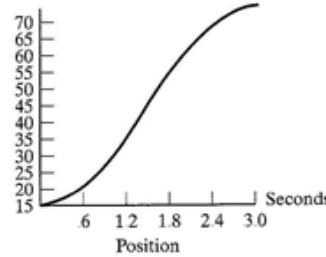
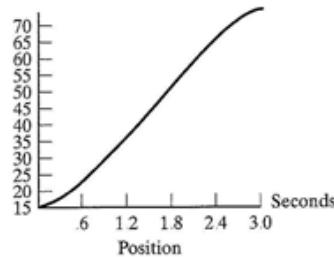
$$v(t) = v_0 + at$$
$$p(t) = p_0 + v_0 t + \frac{1}{2} a t^2$$



속도 계획

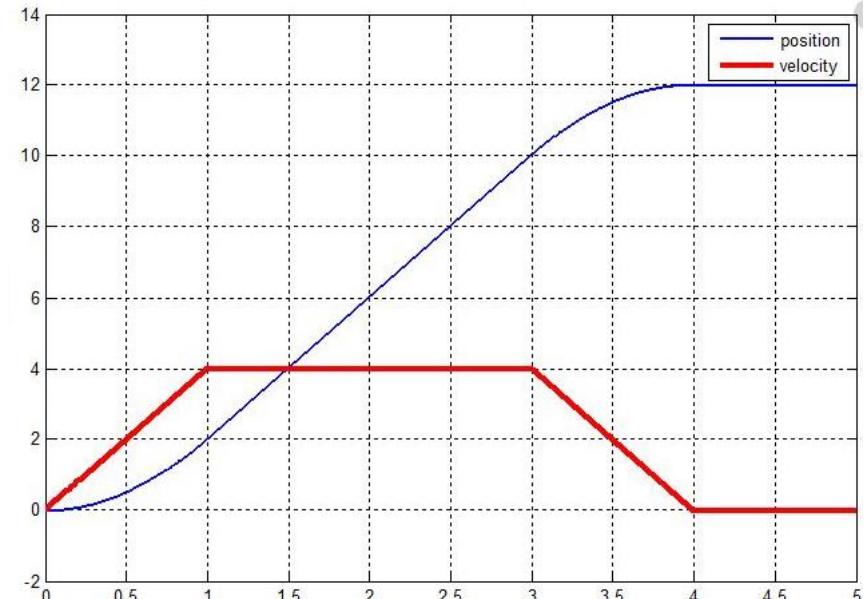
▶ 사다리꼴 속도 계획법

- 다음과 같이 목표에 맞게 각 구간별로 위치, 시간, 가속도를 구할 수 있다.
- 가속도에 대한 제약을 준다면, 승차감까지 고려할 수 있다.



(a)

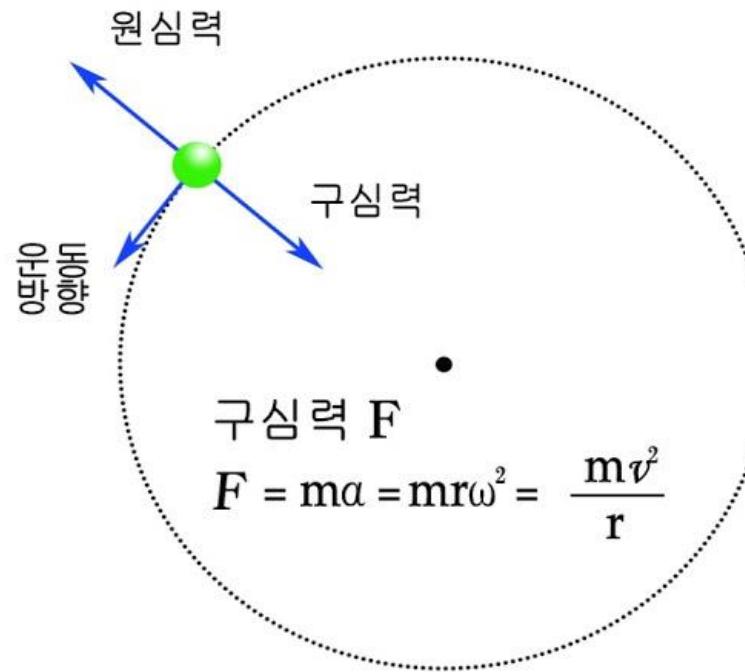
(b)



속도 계획

▶ 경로 기반 속도 계획

- 차량 외부 요인(도로, 노면, 날씨, 제한속도) 중 도로 모양만을 고려해서 속도 계획을 한다.
- 곡률을 구해서 곡률 기반에서 차량이 주행할 수 있는 최대 속도를 계산
- 차량은 곡선에서 주행할 때 원심력을 받아서, 곡선에서는 속도를 낮춰야 한다. 따라서 속도 계획을 통해 차량이 전복되는 것을 방지하고, 직선에서는 최대 속도로 주행하며, 곡선에서는 안정적인 속도로 주행할 수 있다.

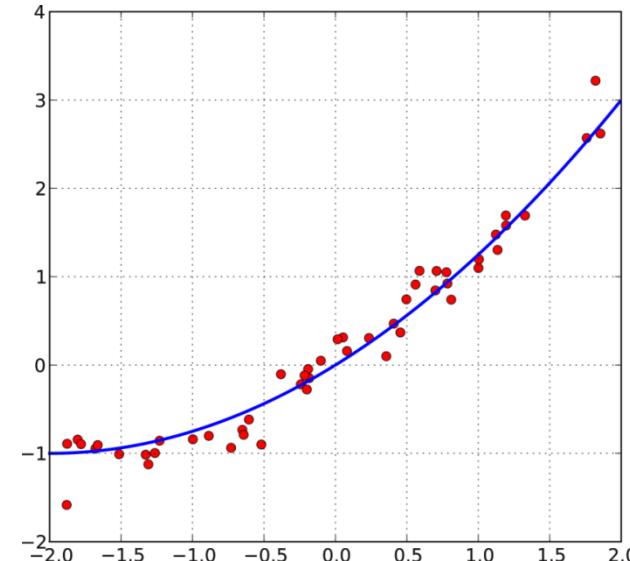
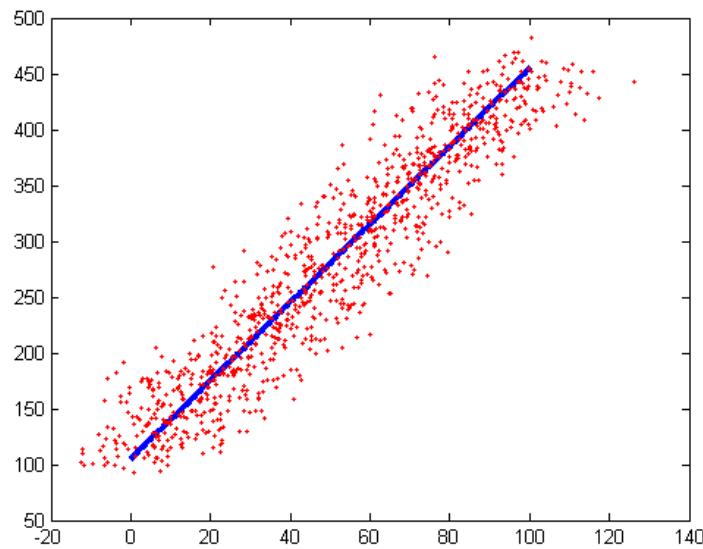


속도 계획

➤ 경로 기반 속도 계획

• 최소자승법

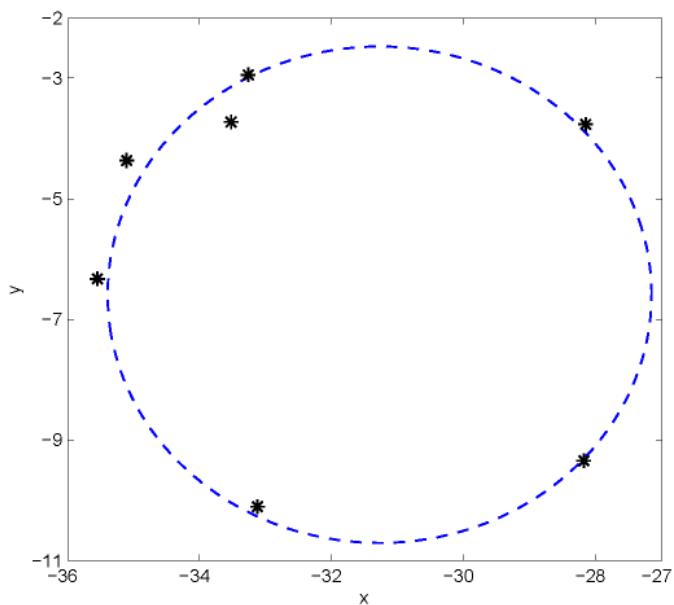
- ✓ 어떤 점들의 분포를 직선이나 곡선으로 근사화하는 방법
- ✓ 수학적 도구, 수치해석, 회귀분석, 영상처리 다양한 분야에서 사용한다.
- ✓ 어떤 모델의 파라미터를 구하는 한 방법으로서, 데이터와 residual의 합을 최소화하도록 모델의 파라미터를 구하는 방법
- ✓ residual은 어떤 데이터가 추정된 모델로부터 얼마나 떨어진 값인가를 나타내는 용어이다.
- ✓ 대수적 방법, 해석학적 방법, 비선형 최소자승법



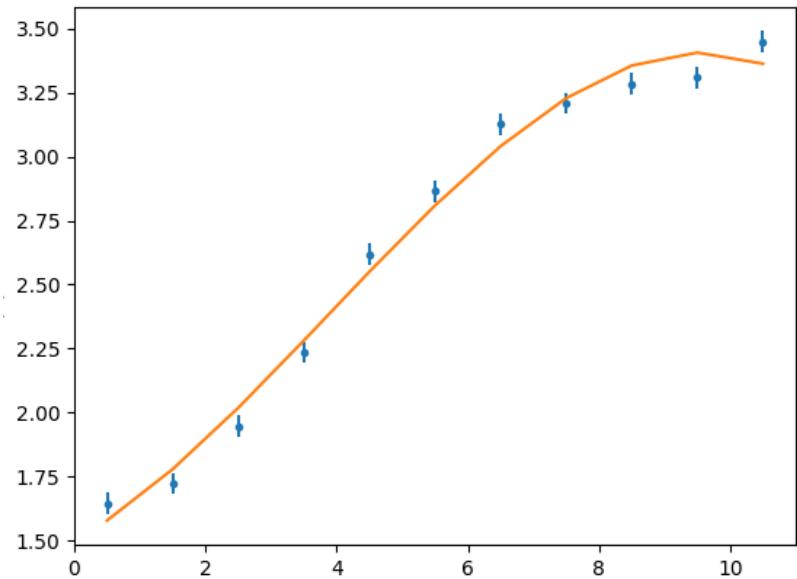
속도 계획

▶ 경로 기반 속도 계획

- 최소자승법



☞ Circle fitting



☞ curve fitting

속도 계획

➤ 경로 기반 속도 계획

- 최소자승법(대수적 방법)

- ✓ 행렬식 형태로 표현한 후에 선형대수학을 적용하는 방법으로 모델 파라미터를 구한다.
- ✓ pseudo inverse 라는 방법을 이용해 계산한다.

$$\begin{array}{l} ax_1 + b = y_1 \\ ax_2 + b = y_2 \\ \vdots \\ ax_n + b = y_n \end{array} \quad \left(\begin{array}{cc} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{array} \right) \left(\begin{array}{c} a \\ b \end{array} \right) = \left(\begin{array}{c} y_1 \\ \vdots \\ y_n \end{array} \right) \quad X = \text{pinv}(A)B \\ \qquad \qquad \qquad AX = B \quad \quad \quad = (A^T A)^{-1} A^T B \end{array}$$

속도 계획

➤ 경로 기반 속도 계획

- 최소자승법(해석학적 방법)

- ✓ 모델 파라미터들로 편미분한 후에 그 결과를 0으로 놓고 연립 방정식을 푸는 방법
- ✓ 다소 복잡함

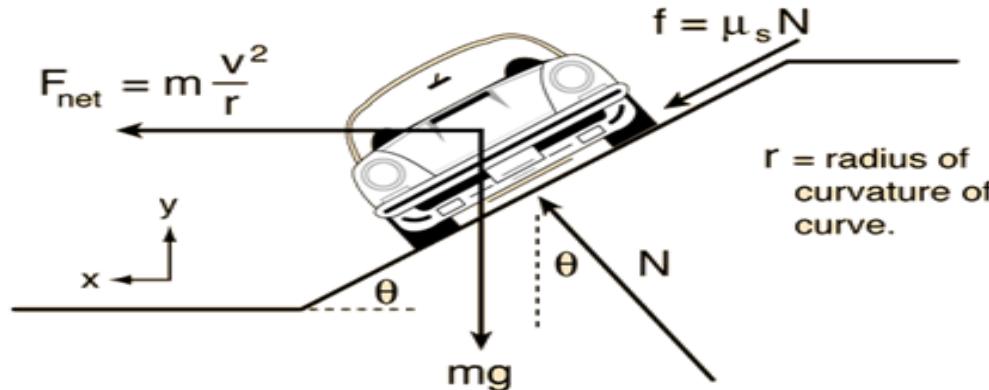
$$\frac{\partial}{\partial a} \sum_{i=1}^n r_i^2 = \sum_{i=1}^n 2(y_i - ax_i - b)(-x_i) = 0$$

$$\frac{\partial}{\partial b} \sum_{i=1}^n r_i^2 = \sum_{i=1}^n 2(y_i - ax_i - b)(-1) = 0$$

속도 계획

▶ 경로 기반 속도 계획

- 도로에서 받는 차량의 힘으로부터 차량이 전복되지 않는 주행가능한 최대속도를 구할 수 있다.
- r은 경로의 곡률반지름, g는 중력, u_s 는 운동 마찰력



Force equations at maximum speed v, at threshold of sliding up incline.

$$\sum F_x = m \frac{v^2}{r} = N \sin \theta + \mu_s N \cos \theta$$

$$\sum F_y = 0 = N \cos \theta - \mu_s N \sin \theta - mg$$

Solving this pair of equations for the maximum speed v gives:

$$v_{max} = \sqrt{\frac{rg(\sin \theta + \mu_s \cos \theta)}{\cos \theta - \mu_s \sin \theta}}$$

The limiting cases are:

$$v_{max} = \sqrt{rg \tan \theta}$$

Frictionless case

$$v_{max} = \sqrt{rg \mu_s}$$

Flat roadway

속도 계획

➤ 경로 기반 속도 계획

- 행렬을 이용해서 쉽게 곡률 반지름 r 을 구할 수 있다.

$$(x-a)^2 + (y-b)^2 = r^2$$

$$x^2 + y^2 - 2ax - 2by + a^2 + b^2 - r^2 = 0$$

$$c = a^2 + b^2 - r^2$$

$$x^2 + y^2 - 2ax - 2by + c = 0 \quad (a, b, c \text{에 대해서 정리})$$

$$-2ax - 2by + c = -x^2 - y^2$$

$$\begin{pmatrix} -2x_1 & -2y_1 & 1 \\ \vdots & \vdots & \vdots \\ -2x_n & -2y_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} -x_1^2 - y_1^2 \\ \vdots \\ -x_n^2 - y_n^2 \end{pmatrix}$$

$$AX = B$$

$$X = (A^T A)^{-1} A^T B$$

$$r = \sqrt{a^2 + b^2 - c}$$

속도 계획

➤ 경로 기반 속도 계획

- 행렬을 이용해서 쉽게 곡률 반지름 r을 구할 수 있다.

```
def curveBasedVelocity(self,global_path):
    out_vel_plan=[]
    for i in range(0,self.point_num):
        out_vel_plan.append(self.car_max_speed)

    for i in range(self.point_num,len(global_path.poses)-self.point_num):
        x_list=[]
        y_list=[]
        for box in range(-self.point_num,self.point_num):
            x=global_path.poses[i+box].pose.position.x
            y=global_path.poses[i+box].pose.position.y
            x_list.append([-2*x,-2*y,1])
            y_list.append(-(x*x)-(y*y))

        x_matrix=np.array(x_list)
        y_matrix=np.array(y_list)
        x_trans=x_matrix.T

        a_matrix=np.linalg.inv(x_trans.dot(x_matrix)).dot(x_trans).dot(y_matrix)
        a=a_matrix[0]
        b=a_matrix[1]
        c=a_matrix[2]
        r=sqrt(a*a+b*b-c)
        v_max=sqrt(r*9.8*self.road_friction)
        if v_max>self.car_max_speed :
            v_max=self.car_max_speed
        out_vel_plan.append(v_max)

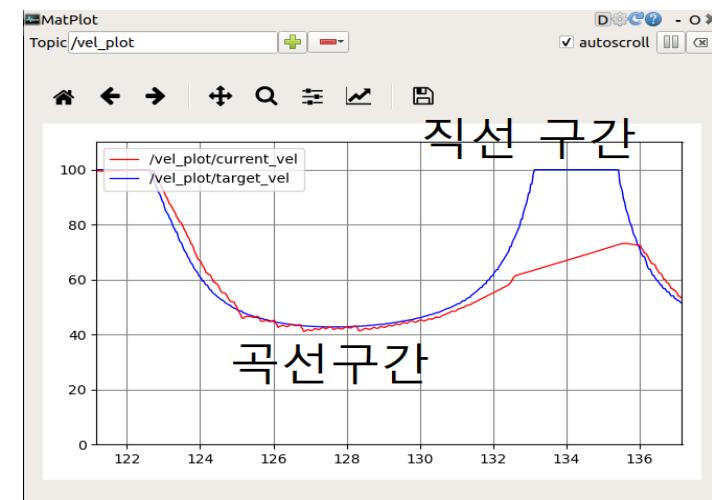
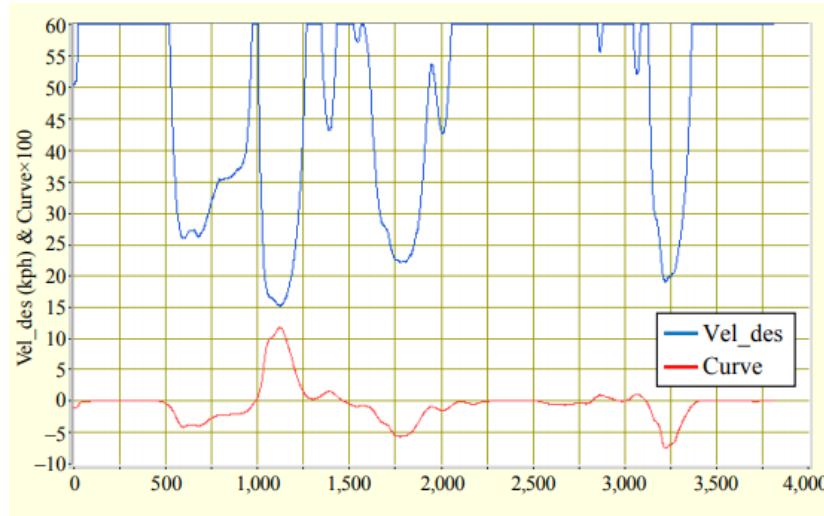
    for i in range(len(global_path.poses)-self.point_num,len(global_path.poses)):
        out_vel_plan.append(0)

    return out_vel_plan
```

속도 계획

▶ 경로 기반 속도 계획

- 곡률에 따른 목표속도를 계산한 결과
- 직선 구간에서는 곡률반지름이 무한대에 가까워지기 때문에 주행할 속도 혹은 차량의 최대 주행속도로 제한을 걸어준다.



속도 계획

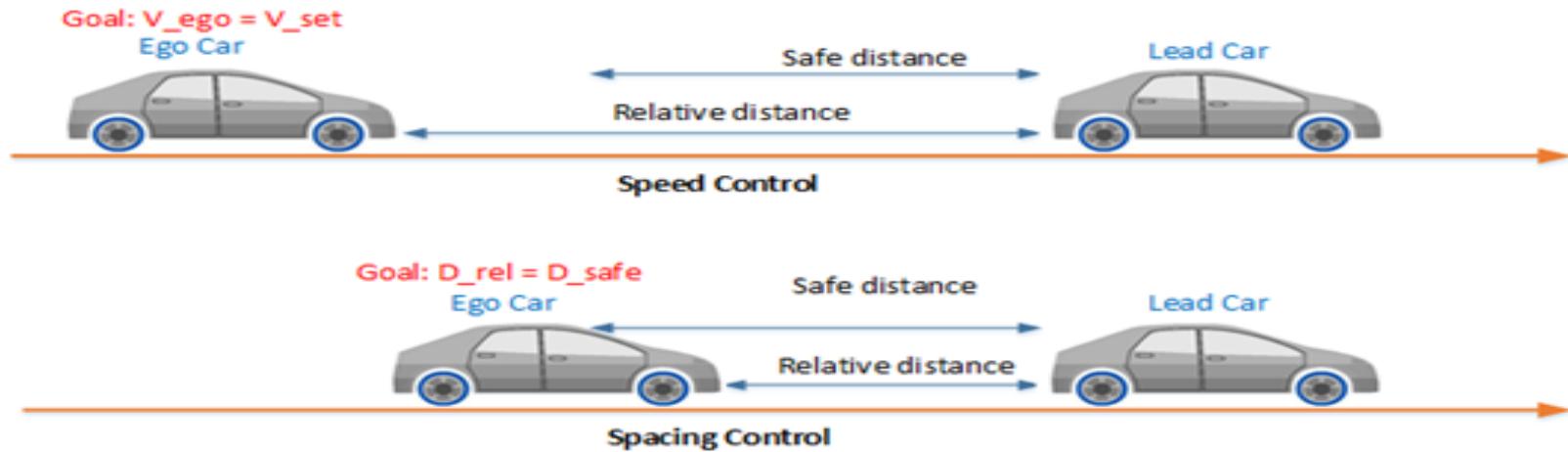
➤ 실습 8 : 경로기반 속도 계획

- 최단 경로 탐색 결과로 얻은 경로에 대해서 각 경로점마다 차량이 주행할 수 있는 최대 속도를 구해라

속도 계획

➤ Adaptive Cruise Control(ACC)

- ACC는 운전자(EgoCar)가 설정한 속도로 주행을 하다가 레이다, 라이다, 카메라 등과 같은 환경인지 센서로 앞 차(LeadCar)를 인지하고 앞 차와의 간격을 유지하는 것을 말합니다.



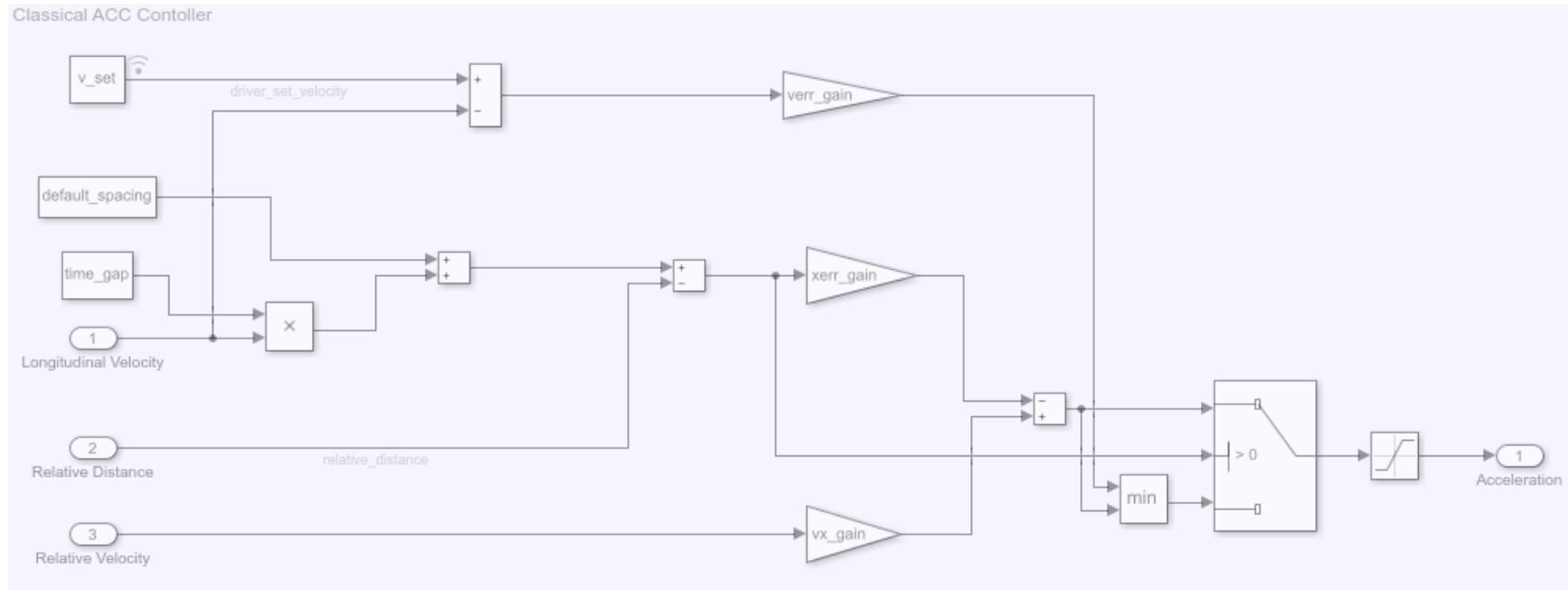
$$D_{safe} = V_{ego} * time_{gap} + defaultSpace$$

$$Acceleration = V_{rel} * gain_{vel} - gain_{dis} * (D_{safe} - D_{rel})$$

속도 계획

➤ Adaptive Cruise Control(ACC)

- ACC는 운전자(EgoCar)가 설정한 속도로 주행을 하다가 레이다, 라이다, 카메라 등과 같은 환경인지 센서로 앞 차(LeadCar)를 인지하고 앞 차와의 간격을 유지하는 것을 말합니다.



<https://kr.mathworks.com/help/mpc/ug/adaptive-cruise-control-with-sensor-fusion.html>

속도 계획

➤ Adaptive Cruise Control(ACC)

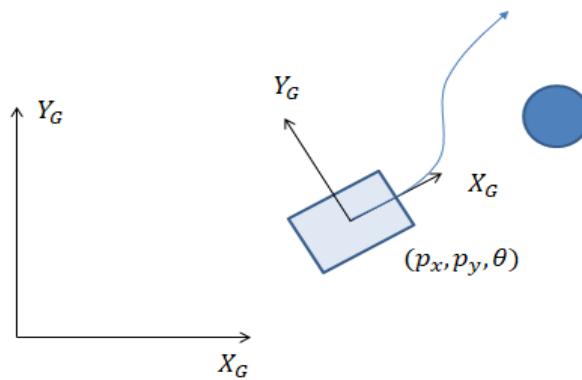
- 주변 장애물에 대한 정확한 위치를 알아야한다.
- 시뮬레이터에서 제공하는 정답 데이터를 사용
 - ✓ 타입 : morai_msgs/ObjectInfo
 - ✓ 토픽 : /Object_topic
- 주변 장애물에 대한 위치(pose)는 Global 좌표계로 나온다.

		6.58KB/s	49.65
▼ [✓] /Object_topic	morai_msgs/ObjectInfo		
header	std_msgs/Header		(179.99948120117188,)
heading	float64[]		1
num_of_objects	int32		(2,)
object_type	int16[]		(14.928829193115234,)
pose_x	float64[]		(1102.7318115234375,)
pose_y	float64[]		(-0.6700925230979919,)
pose_z	float64[]		(0.75,)
size_x	float64[]		(0.75,)
size_y	float64[]		(0.4300000071525574,)
size_z	float64[]		(0.0,)
velocity	float64[]		

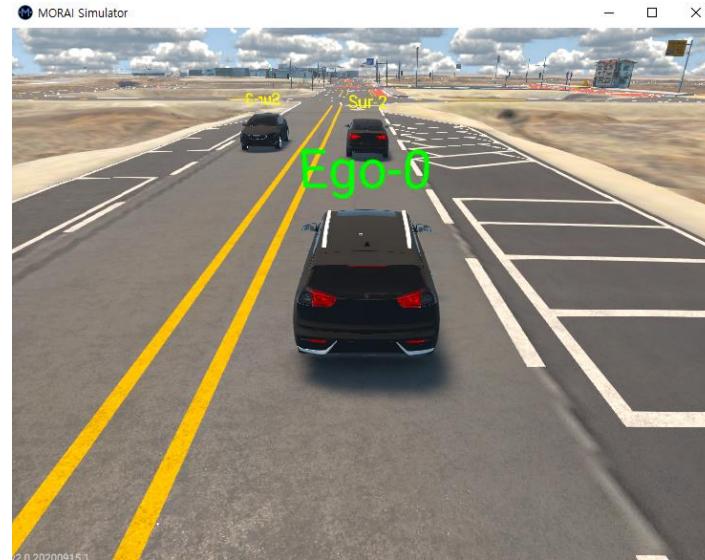
속도 계획

➤ Adaptive Cruise Control(ACC)

- 주변 장애물을 로봇 좌표계(base_link)로 변환한다.
- 변환은 Translation & Rotation Transformation Matrix O|U용
- 차량 뒤쪽에 있는 장애물들은 고려하지 않는다.



$$\begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix}_G = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ 1 \end{bmatrix}_L$$



<https://kr.mathworks.com/help/mpc/ug/adaptive-cruise-control-with-sensor-fusion.html>

속도 계획

➤ 실습 9 : ACC 구현

- 전방 차량 (Lead Car)를 선택
 - ✓ 지역 경로의 일정 범위(장애물의 크기 고려) 안에 있는지 체크
 - ✓ 그 중 가장 가까운 장애물을 선택
- ACC 알고리즘을 적용해서 Safe Distance를 유지할 수 있는 가속/감속 값을 계산한다.