[양식 4] 자율주행자동차 기술보고서

국제 대학생 창작자동차 경진대회

자율주행자동차 기술보고서

■ 자동차번호 :_____128_____

■ 참 가 팀 명 :_____항대카비스_____

■ 참가대학명 :_____한국항공대학교_____

■ 주요내용

- 자율주행자동차 아키텍처
  - ▸ 플렛폼과 센서의 Layout
  - ▸ 센서 장착 설계
- 자율주행자동차 제어 로직 개발
  - ▸ 환경 인지 로직
  - ▸ 판단 로직
  - ▸ 플랫폼 제어 로직

※ 1. 제어로직 및 주요장치에 대해서는 전년도 출전자동차와의 차이점에 대하여 기술되어 있어야 한다.
2. 작성형식은 자유이며 표지를 포함하여 총 30페이지 이내
(휴먼명조체로 크기는 12포인트, 줄간격은 160%로 작성)

■ 첨부

- 자율주행자동차 제원표 1부 (양식 5)

2022.08.19.

팀 팀장 변정민 (서명)

국제 대학생 창작자동차 경진대회 조직위원장 귀하

[양식 5] 자율주행자동차 제원표

국제 대학생 창작자동차 경진대회

자율주행자동차 제원표

■ 자동차번호 :_____128_____

■ 참 가 팀 명 :_____항대카비스_____

■ 참가대학명 :_____한국항공대학교_____

▸ 일반제원

구분	제원	보기
플랫폼 중량	kgf	
플랫폼 최대 길이	1,600(mm)	
플랫폼 최대 폭	1,160(mm)	
플랫폼 최대높이	1,250(mm)	
기타	13-inch Tire	금호 Solus TA21 175/60R13

▸ 자율주행자동차 제원

구분	제원	보기
LiDAR	16ch, 5~20Hz, 0.3Mpps	Velodyne Puck VLP-16
Camera	720p, 90fps(depth) 1080p, 30fps(RGB)	Intel Realsense D435
IMU	3-axis accelerometers, 3-axis gyros, 3-axis magnetometers	VectorNav VN-100
GPS	RTK up to 20Hz, 0.01m+1ppm CEP, 수렴시간 <10s	u-blox ZEP-F9P
Computing Unit	4core 8thread 3.40GHz 8GB RAM	Intel i7-6700

2022.08.19.

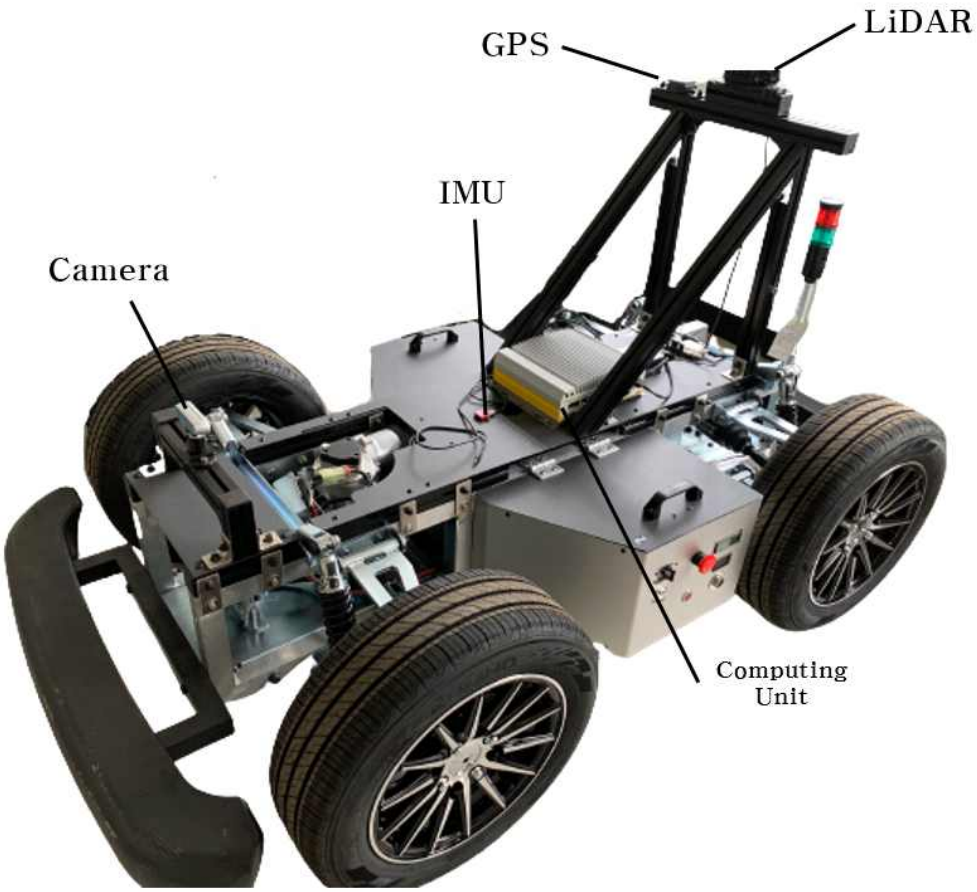
팀 팀장 변정민 (서명)

국제 대학생 창작자동차 경진대회 조직위원장 귀하

목차

I. 자율주행자동차 아키텍처

1. 플랫폼과 센서의 Layout



< 플랫폼과 센서 Layout>

사용된 부품의 이름과 간단한 제원은 다음과 같다.

* ERP-42
  - 길이×전폭×높이 : 1,600×1,160×1240mm
  - 지상고: 110mm, 윤거: 985mm, 축거: 1,040mm
  - 모터: 3kW AC모터, 3,000rpm(rate), 6,000rpm(peak), 9.55N-m(rate), 45N-m(peak)
  - 배터리: Li-ion 48V-40Ah, 2,417Wh
  - 컨트롤러: 32bit RISC MCU, 12V In, 5V Out
  - 금호타이어 Solus TA21 175/60R13

* Camera - Intel Realsense D435
    - Depth 1280×720, 90fps, 87° ×58° FOV
    - RGB 1920×1080, 30fps, 69° ×42° FOV
* 3D LiDAR - Velodyne Puck VLP-16
    - 16 channels, Rotation rate 5~20Hz, 0.3 million points/second
* IMU - VectorNav VN-100
    - 3-axis accelerometers, 3-axis gyros, 3-axis magnetometers
* GPS - u-blox ZED-F9P
    - GNSS, RTK up to 20Hz, 0.01m + 1ppm CEP, 수렴시간 <10s
* Computing Unit
    - Intel i7-6700, 8GB RAM

2. 센서 장착 설계

2.1) Camera / LiDAR

Camera와 LiDAR의 경우 시야를 가리지 않는 것이 중요하다. 따라서 카메라를 높게 배치한 후 LiDAR를 낮게 배치하게 되면, 카메라에서는 좋은 영상을 얻을 수 있지만, 3D LiDAR를 사용하기 때문에 카메라와 카메라 지지대의 pointcloud가 형성되어 정보를 얻는데 제약이 생기고, LiDAR data를 처리함에 있어 어려움이 생기게 된다. 따라서 카메라를 앞쪽 낮은 지지대에 설치하고, LiDAR를 후면의 높은 지지대 위에 설치하였다.

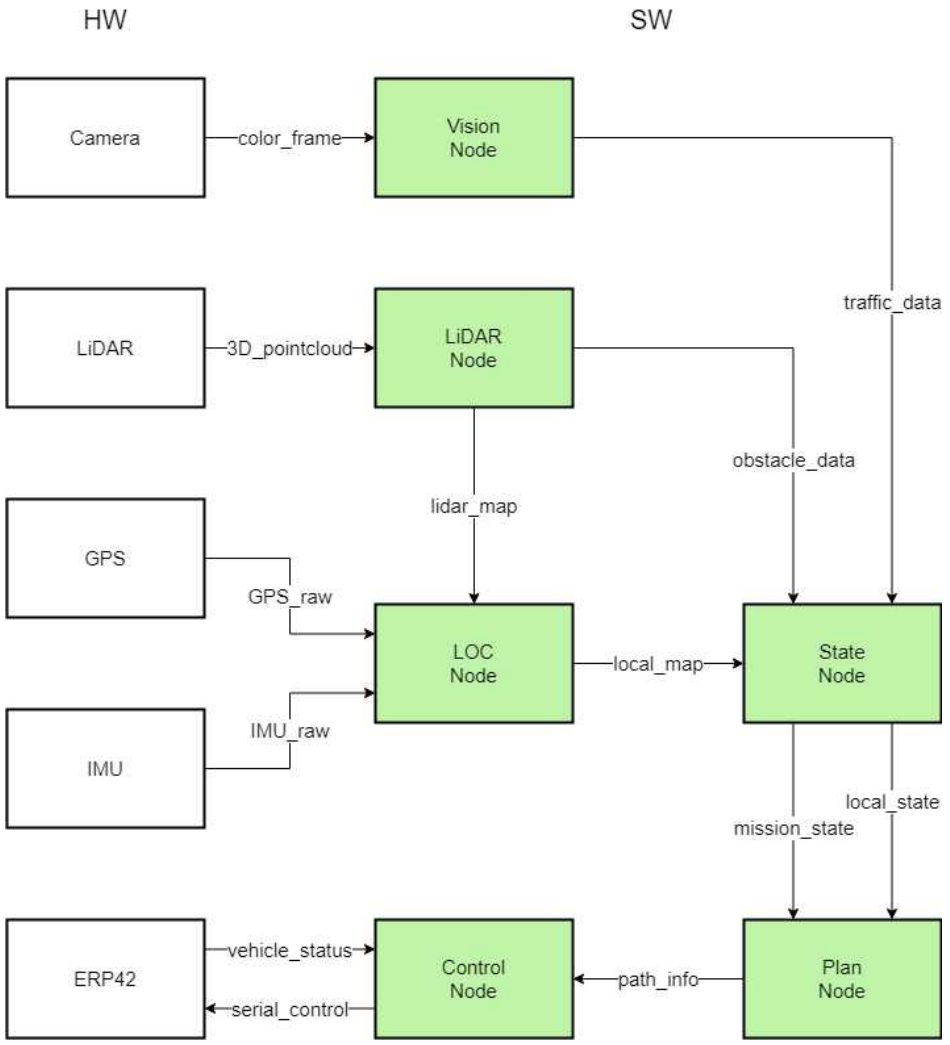
2.2) IMU

IMU는 차량의 운동 상태를 확인할 수 있는 장치이므로, 차량의 무게중심에 IMU를 위치하게 하면 차량의 상태를 정확하게 처리할 수 있어 계산의 복잡함을 줄일 수 있다. 따라서 IMU의 경우 차량 모터와 배터리 위 부분에 설치하였다.

2.3) GPS

GPS의 경우 위성의 신호를 받아야하기 때문에 차량 내부에 위치시키게 되면 간섭이 발생하거나 오차가 발생할 수 있다. 따라서 차량의 가장 높은 곳인 LiDAR 지지대 위에 배치하여 신호를 양호하게 받을 수 있게 하였다.

Ⅱ. 자율주행자동차 제어 로직 개발



< 자율주행자동차 구성 >

본 대회에서 사용하는 자율주행자동차는 카메라, LiDAR, GPS와 IMU를 장착하여 사용하였으며, 내부 구조는 위와 같다. 환경 인식 부분은 Vision Node와 LiDAR Node, LOC Node, State Node로, 판단 부분은 Plan Node로 통합되어 있으며, 제어 부분은 Control Node로 이루어져 있다.

1. 환경 인지 로직

차량에 장착되어 있는 카메라, LiDAR, GPS, IMU를 통해 차량 주위의 환경을 인지할 수 있다. 카메라는 차선, 정지선, 표지판, 신호등과 장애물을 인식하는 데 쓰인다. LiDAR는 카메라가 인식하지 못하는 사각지대의 장애물을 카메라와 더불어 인식하며, SLAM을 이용하여 local 지도를 작성하는 데 사용

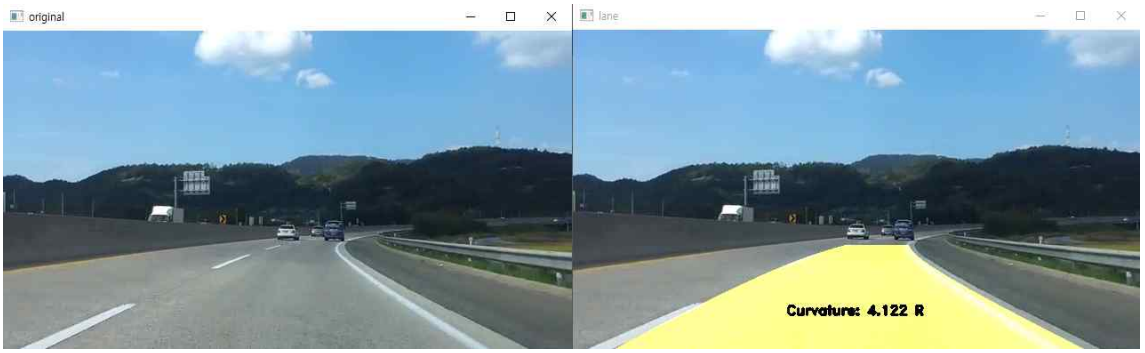
한다. GPS와 IMU는 현재 위치를 추정하고, SLAM을 이용하여 지도를 작성하며, 작성된 지도에서 현재 위치를 확인 및 보정하는데 사용된다.

1.1) Vision Node

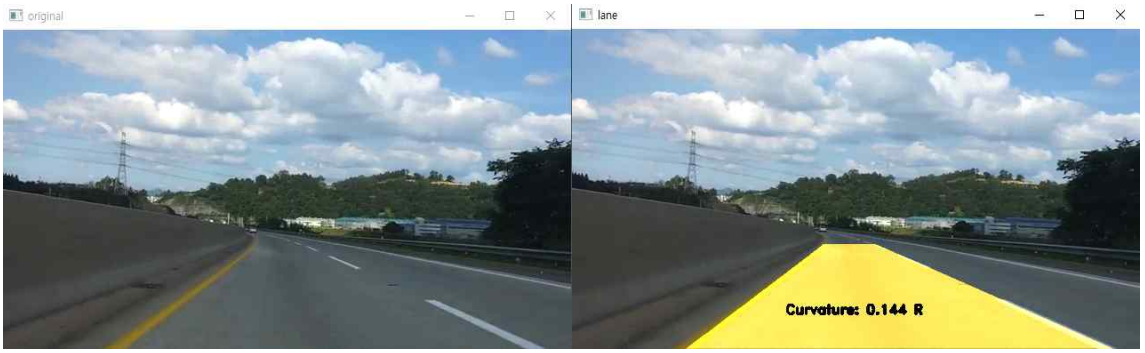
Vision Node에서는 카메라에서 받아온 영상을 처리하여, State Node로 신호등, 정지선, 차선 등의 traffic_data를 전달하는 역할을 한다. 차전 검출 알고리즘은 sliding window를 사용한 방식을 사용하였으며, 표지판, 신호등 등 객체 인식은 yolov5 모델을 이용하였다. 본 차량에는 전방에 Intel RealSense D435 Depth Camera가 장착되어 있으나, 인지 단계에서는 카메라의 color frame만 사용하였다.

1.1.1) 차선 검출 알고리즘

Lane Detection 단계에서는 차량이 현재 주행하고 있는 차선을 인식한다. 먼저 카메라로 인식된 color frame을 HLS 영역으로 변환시킨 후 Sobel filter를 적용한다. Filtering 된 영상을 임계값으로 이진화한 후 설정된 ROI를 이용하여 warpping을 하여 bird-eye view로 전환한다. 그 후 전환된 영상으로 sliding window를 이용하여 차선을 검출하고 검출된 차선의 곡률을 계산한다.



< 차선 검출 1 >



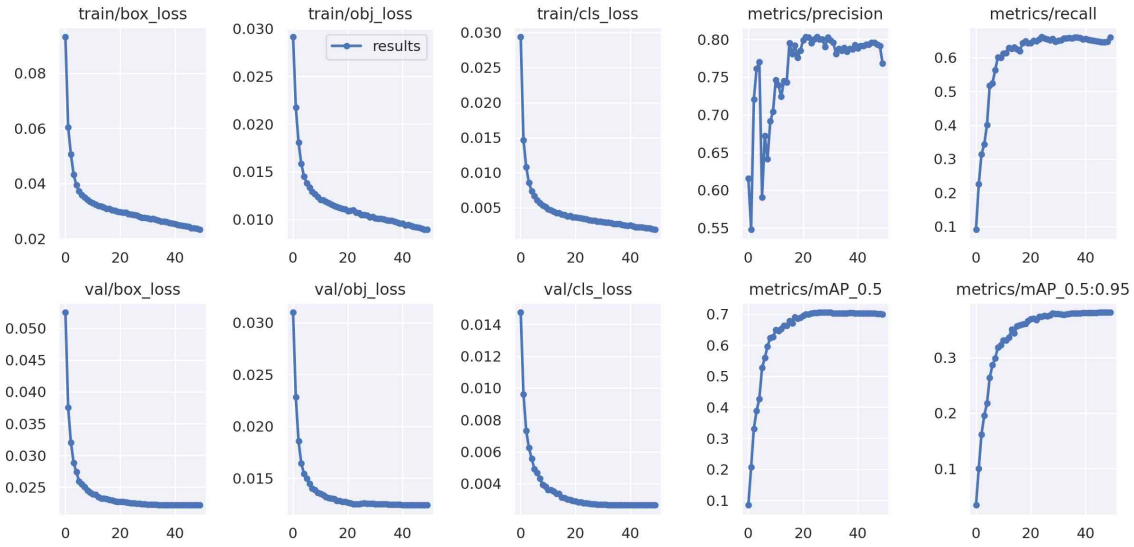
< 차선 검출 2 >

< 차선 검출 3 >

시뮬레이션과 영상을 통해 차선을 검출한 결과, 같은 차선을 따라갈 때에는 좋은 성능을 보였으나, 주행중인 본 차선을 벗어나야 하거나 교차로를 지날 때와 같은, 주위 차선을 모두 인식하여야 하는 상황에서는 주위 차선은 잘 인식하지 못하였다. 따라서 lanedet을 이용하여 딥러닝을 활용한 차선인식 시스템으로 교체를 시도하고 있다.

1.1.2) 객체 인식

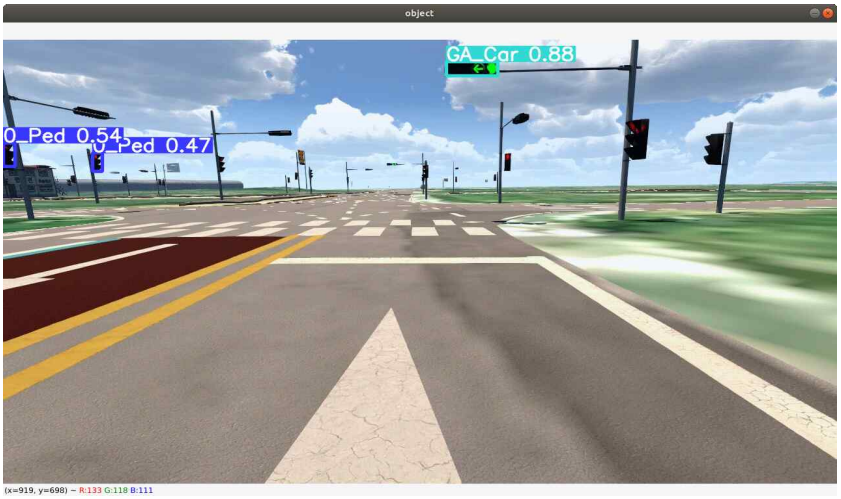
표지판과 신호등 인식은 yolov5 모델을 이용하여 학습한 가중치를 바탕으로 분류하였다. 학습된 dataset은 AIHUB의 "차선/횡단보도 인지 영상(수도권 외)" 와 "신호등/도로표지판 인지 영상(수도권 외)" 를 사용하였다.



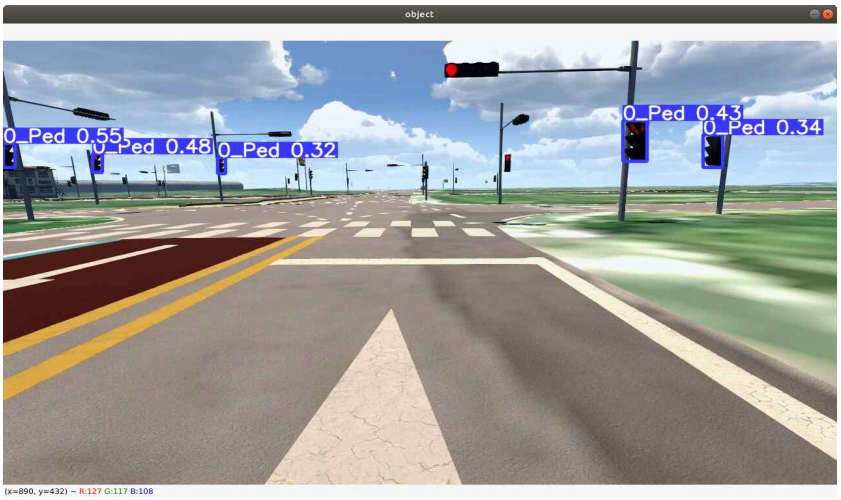
< 사진 : yolov5 result >

학습 결과를 보면 recall 성능이 좋지 않은 것을 확인할 수 있었다. 다음은 MORAI 시뮬레이션 상에서 획득한 이미지를 통하여 객체 인식을 실행한 결과이다.

< 객체 인식 : 신호등 1 >



< 객체 인식 : 신호등 2 >

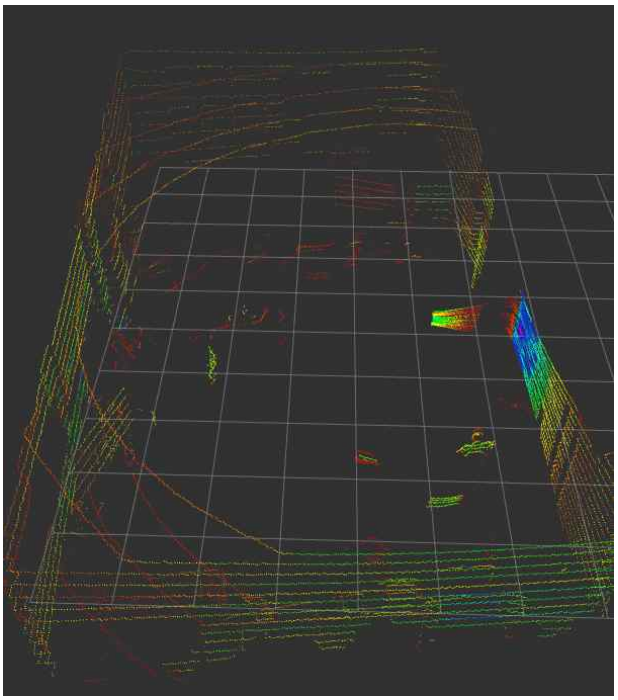


< 객체 인식 : 신호등 3 >

결과를 보면 녹색등은 잘 인식하는데 반해 적색 신호를 잘 인식하지 못하는 것을 확인할 수 있었다. 성능 개선을 위하여 dataset과 라벨 검증을 진행한 결과, 라벨의 개수가 “신호등/도로표지판 인지 영상(수도권 외)” 의 경우 의미 없는 분류에 대해 필터링을 거쳤음에도 불구하고, 168개의 라벨로 분류되어 있었으며, 오분류된 라벨도 다수 존재하는 것을 확인할 수 있었다. 따라서 dataset 교체를 통해 성능 향상을 목표로 하고 있다.

1.2) LiDAR Node

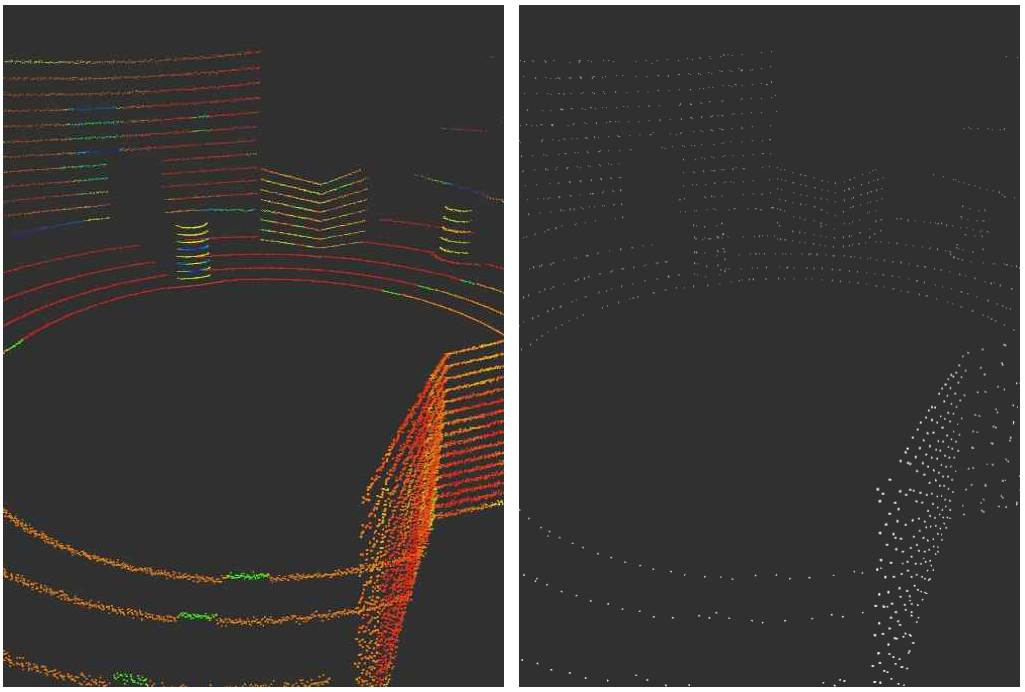
LiDAR Node는 LiDAR에서 받아온 3D point cloud를 처리한다. 데이터 처리의 복잡도를 줄이기 위하여 point cloud를 Voxel Grid downsampling하여 데이터를 간소화시킨 후, RANSAC을 이용하여 객체와 도로를 분리하는 작업을 거친다. 분리된 객체는 DBSCAN 알고리즘을 이용하여, 분류한 후, State Node에는 객체 장애물 정보를, LOC Node에는 처리된 3D map을 전달한다.



< LiDAR로 획득한 3D point cloud >

1.2.1) Voxel Grid Downsampling

Voxel Grid Downsampling은 단위 크기(leaf size)의 3차원 공간 내의 존재하는 point를 한 point로 줄여 간소화하는 방법이다. Voxel Grid Downsampling을 이용하면, LiDAR를 통해 획득한 3D point cloud를 단순하고 빠르게 처리할 수 있다.



< Voxel sampling 전 / Voxel sampling 후 >

1.2.2) RANSAC

외부에서 사용하는 자율주행 LiDAR의 특성 상 획득한 point cloud는 지면(도로)과 객체가 이어져 있을 수밖에 없다. 이 때 지면을 따로 분류할 수 있다면 객체들은 서로 분리되게 되어 쉽게 객체를 인식할 수 있다. 따라서 지면에서 검출된 point는 따로 분류하여 제거하여야 한다. 지면을 구분하기 위하여, 유사한 normal vector를 가진 point를 clustering 하여야 하는데 이때 RANSAC을 이용하여 분류할 수 있다.

1.2.3) DBSCAN

DBSCAN은 밀도 기반 군집화 알고리즘으로 분리된 객체 데이터를 군집화하기 위해 사용된다. 밀도가 높아 cluster의 중심이 되는 core point와 cluster의 경계가 되는 border point로 cluster의 경계를 도출하여, cluster와 noise를 구분한 후 각각의 cluster를 객체로 인식하게 된다.

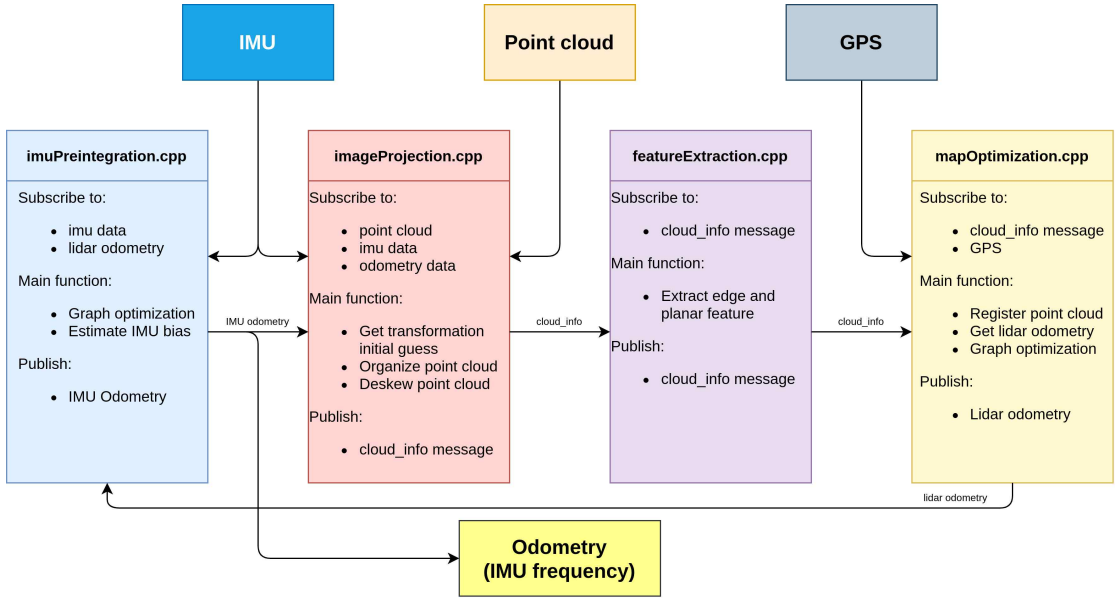
1.3) LOC Node

LOC Node에서는 GPS와 IMU에서 받아온 raw data와 LiDAR Node에서 받아온 3D map을 통하여 위치 정보를 처리하여 State Node에 현재 위치와 local

map을 전달한다.

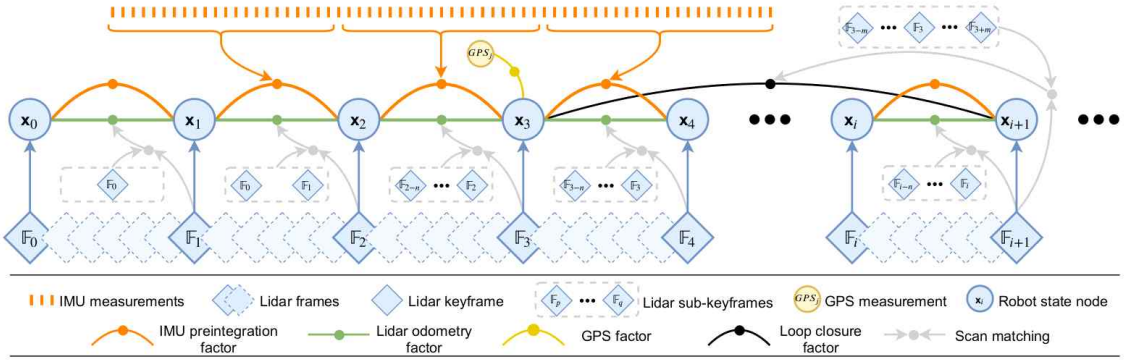
1.3.1) LIO SAM

자율주행에 있어서 장애물을 피하는 것은 물론 잘 피하기 위해서는 현재의 위치를 정확하게 파악하는 것이 매우 중요하다. SLAM을 사용할 때, 순차적으로 움직임을 추정하는 과정에서 오차가 발생하게 되고 이것이 누적되면서 실제 값과는 상당한 편차가 발생하게 된다. 이러한 문제를 해결하기 위하여, LiDAR뿐 아니라 IMU 센서를 같이 사용하여 오차를 예측이 가능한 LIO SAM 기반 SLAM을 사용하였다. 관성 측정 장치인 IMU를 통합해 추정된 움직임은 point cloud의 왜곡을 제거하고, 주행 거리 측정 최적화를 위한 초기 데이터를 생성해 IMU의 편향을 보정한다.



< LIO SAM Architecture>

LIO SAM은 Raw IMU 측정값을 이용하여 LiDAR frame 간의 nonlinear motion model을 가정하고 이를 이용해 LiDAR odometry optimization을 위한 초기값을 설정한다. 그리고 초기값을 이용해 IMU의 편향를 추정하고, global factor graph를 도입해 Global map에서 scan matching이 아닌 이전 LiDAR scan을 marginalize하여 자세 최적화를 수행한다.



< Factor graph >

IMU를 통해 frame의 시작과 끝 사이의 IMU 데이터를 사용해 회전각을 계산하고 IMU Preintegration에서 데이터를 사용하여 변환각을 계산한 다음에 motion 왜곡에 대한 보정을 수행한다. IMU Preintegration은 간략하게 IMU data로 기인하는 모든 측정값을 factor graph에 추가하기 전에 단 하나의 factor로 integration하는 방법이다. Motion 왜곡 보정 후에 현재 frame의 point cloud에 대한 각 frame의 curvature를 계산한 후 feature Extraction한다. Scan-to-map matching을 통해 feature extraction하고 scan-to map을 반복해 map optimization을 수행한다. 이러한 기술들의 작용으로 Pose graph optimization 통해서 보다 정밀한 LiDAR 제어를 할 수 있다.

2. 판단 로직

판단 부분은 State Node와 Plan Node로 이루어져 있으며, State Node는 인식된 정보를 바탕으로 현재 상태와 미션 정보를 도출하며, Plan Node는 도출된 상태를 이용하여 자동차가 이동하여야 하는 경로를 생성하는 역할을 한다. State Node는 Vision Node와 LiDAR Node, LOC Node에서 각각 교통 정보와 장애물 정보, 로컬 맵을 받아 Plan Node에 로컬 상태와 미션 상태를 전송한다. Plan Node는 State Node에서 받은 정보를 통해 경로를 수립하고 수립된 경로를 Control Node에 전달한다.

2.1) State Node

State Node에서는 Vision Node, LiDAR Node, LOC Node에서 받은 데이터를 통합하여 현재 위치, 주변 상태를 처리한다. 인식된 정보를 바탕으로 수행하여야 하는 미션을 분류하고, 현재 상태와 미션 수행 상태를 Plan Node에 전달하는 역할을 한다. 미션 인식은 Vision Node에서 표지판 인식 결과와

bounding box를 통한 표지판 크기를 통해 미션 진입을 인식하고, LiDAR로 습득한 point cloud의 거리 정보를 통해 재차 검증하는 절차를 거친다.

2.2) Plan Node

Plan Node에서는 State Node에서 보낸 mission 정보와 local state을 통하여 path planning을 실시한다. 이전에 이미 수립된 경로를 바탕으로 실시간으로 새로운 경로를 갱신한다. A* 알고리즘을 이용하여 현재 상태와 이동하여야 하는 경로를 Control Node에 전달한다.

2.2.1) A* 알고리즘

경로 탐색 알고리즘으로는 A* 알고리즘을 적용하였다. A* 알고리즘은 RRT 알고리즘에 비해 성능적으론 큰 차이가 발생하지 않는 반면 탐색에 걸리는 시간 이득이 발생하기 때문이다. A* algorithm은 start node만을 설정하고 다른 모든 node에 대한 최단 경로를 파악하는 다른 알고리즘과는 달리 start node와 destination node를 명확하게 지정해 이 두 node간의 최단 경로를 파악하기에 dead-time이 작다. A* 알고리즘은 heuristics estimates를 통해 algorithm을 개선할 수 있다. heuristics estimates의 제공 방식에 따라 최단 경로의 파악 시간이 결정된다. 최단 경로를 파악할 때 node 사이를 이동할 때 발생하는 cost를 토대로 최단 경로를 파악한다.

$$f(n) = g(n) + h(n)$$

위의 식에서 g(n)은 start node에서 현재 node까지의 비용을 뜻하고, h(n)은 현재 node에서 destination node까지의 예상 cost를 뜻한다. 이를 더한 값이 f(n)이 최소가 되는 node를 다음 node로 선정하는 방식이다. 이때 h(n)이 heuristics function이며 이를 설계하는 방법에 따라 algorithm의 성능이 결정된다. Start node를 Openlist에 넣고 Openlist에 있는 node중에서 1개의 node를 빼서 경로 탐색을 거쳐 최단 경로를 찾는다. 다음 node로의 이동이 끝나면 Openlist에서 뺀 node를 Closelist에 넣는다. 이를 반복하며 start node에서 destination node까지의 최단 경로를 찾는다.

2.3) 미션 수행 알고리즘

2.3.1) 사거리 교차로 / 비신호 회전

사거리 표지판을 카메라로 인식하면 인식된 표지판과 미션 내용에 따라 미리 차선 변경을 수행한다. 사거리에 접근 후 신호등의 크기 및 위치 정보를

Vision Node를 통해 얻어 정지 또는 진행한다. 하나의 신호등을 지나는 동안 여러 장의 영상에서 서로 다른 크기의 신호등이 검출되는 경우가 많았는데, 신호판단 기준으로 사용되는 신호 정보는 가장 근접하여 가장 크기 정보가 큰 신호등을 사용하면서 해결하였다. 사거리에서 좌화전 주행하는 경우에 많은 라인들로 된 차선에 의해 차선이 잘 인식되지 않을 경우를 대비해 GPS를 이용하여 경로 주행하는 알고리즘을 적용했다.

2.3.2) 스쿨존 / 과속방지턱 / 횡단보도 일시정지 / 버스 전용차로

카메라를 통하여 과속방지턱이나 스쿨존 표지판, 횡단보도를 검출하거나, 정지선을 검출하면 먼저 신호를 확인 후 감속한다. 과속방지턱은 일정 시간 후 속도를 되돌리며, 스쿨존의 경우 스쿨존 해제 표지판을 검출한 후 일정시간 후 속도를 되돌린다. 횡단보도의 경우, 감속하거나 정지하고, 신호등 인식 후 청신호를 확인하면, 마지막으로 보행자를 확인하고 재출발한다. 버스 전용차로를 인식하면 최우측 차선으로 이동하도록 계획하였다.

2.3.3) 정적 장애물 회피 / 돌발 장애물 인식 및 일시정지

LiDAR를 통해 정적 장애물을 서서히 인식한 경우, A* 알고리즘을 통해 node들 사이의 cost 계산을 통해 경로를 찾아내 정적 장애물을 피한다. 일정 임계값 이상의 속도로 접근하는 돌발 장애물의 경우 회피 알고리즘을 사용하지 않고 일단 최대로 감속한 후 장애물이 사라지면 진행하도록 하였다.

2.3.4) 주차 / 배달 서비스

배달: 배달 표지판이 인식되면, 배달 목표 지점 인식까지 이동 후, 배달 목표 지점을 인식한 경우 먼저 카메라를 이용하여 접근한 뒤, LiDAR를 이용하여 목표 지점 표지판이 자동차 정 우측에 오도록 하여 5초간 정차한다.

주차: 카메라로 주차 표지판을 읽고 서행 후 라이다 센서를 활용해 주차 공간에 객체가 있는지 확인하고 있다면 직진한다. LIO SAM을 통해 주변 장애물을 인식하고 A* 알고리즘으로 주차 자리에 대한 경로를 생성하여 진행하였다. 주차 종료 판단 이후 직진한 후에 후진을 위해 정지할 거리는 주차할 공간 양옆에 임의의 차량이 있다고 가정하고 차량의 크기를 기준으로 거리를 지정했다.

3. 플랫폼 제어 로직

제어 부분은 Control Node로 이루어져 있으며, Control Node에서는 Plan

Node를 통해 받아온 상태와 경로를 통하여 차량을 제어하는 역할을 한다.

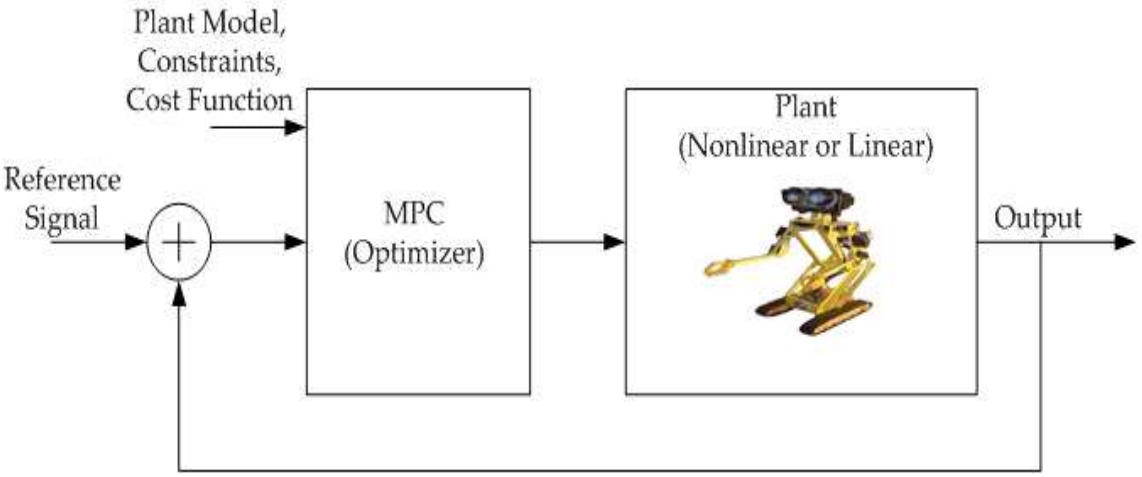
3.1) Control Node

Control Node에서는 Plan Node에서 받아온 현재 상태와 이동할 경로를 통해 실질적으로 ERP42에 조향, 가속, 감속 등의 신호를 serial(RS-232) 통신을 통하여 차량으로 보낸다. 제어는 MPC를 통하여 수행하였다.

3.1.1) MPC

자율주행차에서 Platform Control Logic을 선정할 때 가장 중요하게 고려해야할 것은 안정적으로 제어하는 것이라고 생각하였다. 따라서 실시간으로 외란를 받아들여 제어하는 feedback system을 통해 부드럽고 안정적으로 제어할 수 있는 Control Logic을 선택하려고 하였다. 자율주행 차량의 위치가 실시간으로 변하면서 원하는 정보뿐만 아니라 외란과 간섭 또한 많이 추출되기에 적응형 system을 통해 불확실하게 변하는 목표점을 실시간으로 모델링하는 적응형 제어을 사용하려고 하였다. 차량의 입출력 신호를 이용해 parameter를 추정하고 controller를 실시간으로 갱신하는 방식으로 작동시키기 위하여 MPC(Model Predictive Control) 기술을 사용하였다. 본 대회의 경우에는 자율주행 환경의 비선형성을 고려하여, 선형 시스템을 작동하는데 주로 사용되는 PID제어와 다르게 비선형 시스템 제어할 수 있는 MPC 로직을 사용하였다.

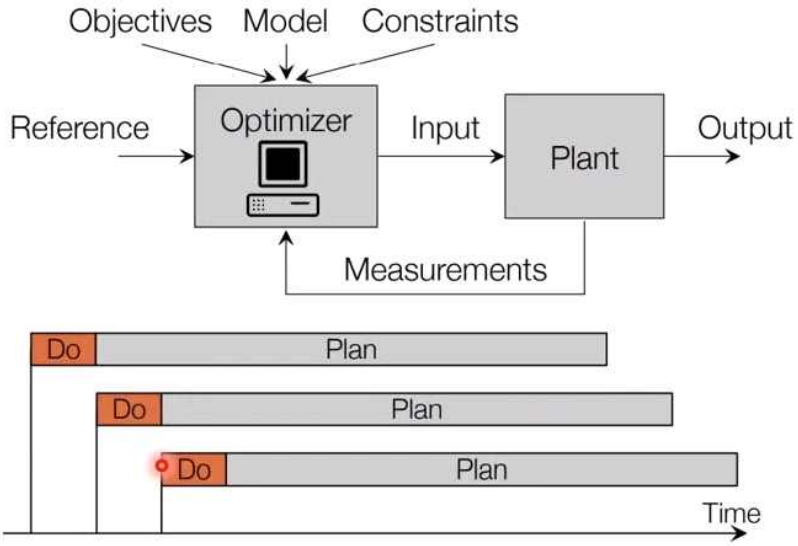
모델예측제어라 불리는 MPC는 제어대상의 model을 이용해 state variable이나 결과를 예측해 이를 바탕으로 최적화을 실행한다. MPC는 optimization process에서 input이나 state variable이 제약 조건을 inequality의 형태로 만족하도록 설계할 수 있다는 것이 장점이다. 그렇기에 특정한 value가 어떤 범위 안에 있도록 설계하는 것이 가능하다.



<Model Predictive Control system>

커브 길을 주행할 때 Camera를 통해 차량은 일정 거리 안에 있는 곡선 차선을 인식하고 Vision Node와 Path Node를 거쳐 차량이 이동해야 할 reference line을 만든다. MPC는 현재의 시간 t를 기준으로, 방향을 틀어 reference line까지 주행해 도착하는 시간 t+n까지 여러 개의 model들을 제시한다. Open-loop system을 이용하면 단순하게 여러 개의 model들을 만들기에 그치지만, MPC의 경우 이후에 급격한 방향 전환이 일어나지 않으면서 reference line에서의 error가 상대적으로 적은 model을 하나 선택해 주행할 수 있다. 이를 반복하면서 발생하는 외란과 간섭을 통해 얻은 오차를 다시 feedback한 후 MPC를 반복하는 closed-loop system을 이용하여 output을 feedback으로 수정을 거치는 것이다.

MPC는 반복되는 feedback과 그에 따른 수정을 거치면서 model은 최상의 결과를 찾아, 차량의 velocity와 acceleration을 조절하면서 오차를 줄이기에 본 차량의 제어 알고리즘 설계 시, 가장 중요하게 고려한 부드럽고 안정적인 automatic control에 가장 잘 어울리는 제어 로직이다.



< MPC Structure>

MPC는 경로를 계산할 plant의 model과 plant가 만족해야 하는 제약 조건을 설정한다. 매 step마다 optimization algorithm을 통해 cost function이 최소가 되는 plan을 계산한다. 계산한 plan의 첫 번째 control action을 실행하고 이를 반복한다. 기존의 PID controller로 multiple input output system을 구축하려면 서로의 input과 output이 영향을 받는 경우가 있기 때문에 구현하기 힘들고 알고리즘 또한 복잡해져 dead-time이 늘어나게 되지만 MPC controller는 그것들을 간소화시켜서 해결해 준다. MPC에서 가장 중요한 것이 현재의 step에서 state variable을 통해 next step의 output을 예측하는 것이기 때문에 dynamic system modeling이 필요하다. Discrete-time state-space model은 다음과 같다.

$$\text{DT state equation} : x(k+1) = A_d(k) + B_d u(k)$$
$$\text{DT output equation} : y(k) = C_d x(k)$$

이를 바탕으로 MPC에서 사용하는 model의 가장 큰 차이점은 Δx를 사용한다는 것이다.

$$\text{state equation} : x(k+1) - x(k) = A_d(x(k) - x(k-1)) + B_d(u(k) - u(k-1))$$
$$\therefore \Delta x(k+1) = A_d \Delta x(k) + B_d \Delta u(k)$$
$$\text{output equation} : y(k+1) - y(k) = C_d \Delta x(k+1) = C_d A_d \Delta x(k) + C_d B_d \Delta u(k)$$
$$\therefore y(k+1) = y(k) + C_d A_d \Delta x(k) + C_d B_d \Delta u(k)$$

위의 형태를 Matrix의 형태로 변환하면 아래와 같은 식을 얻게 된다.

$$x(k) = \left[\Delta x(k)^T y(k)\right]^T$$
$$\begin{bmatrix} \Delta x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A_d & 0 \\ C_d A_d & 1 \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \end{bmatrix} + \begin{bmatrix} B_d \\ C_d B_d \end{bmatrix} \Delta u(k)$$
$$y(k) = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \end{bmatrix}$$

위의 Matrix form이 MPC를 시작하기 위한 기본 model이다. state variable이 x가 아닌 Δx이기 때문에 실시간으로 변하는 target에서 feedback을 반복하며 경로를 예측하여 제어할 수 있다.