programmers

자율주행데브코스

프로젝트 설명

Programmers Lecturer

김기훈

rlgns4861@gmail.com

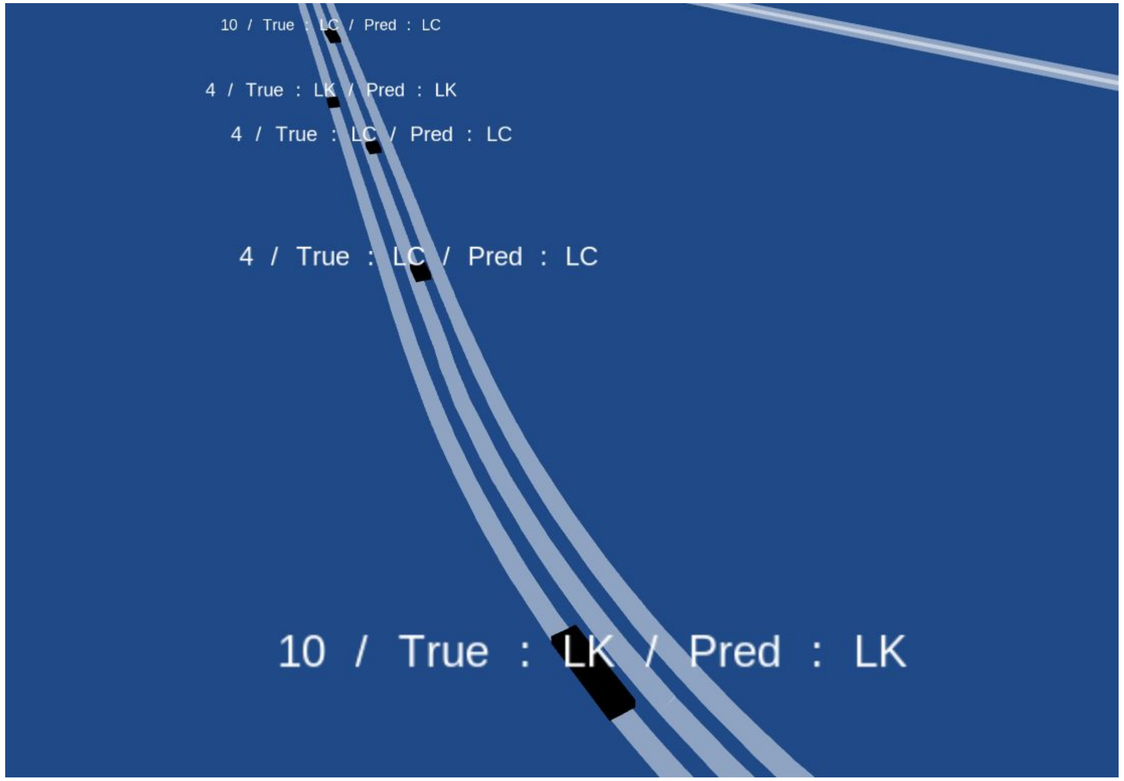
programmers

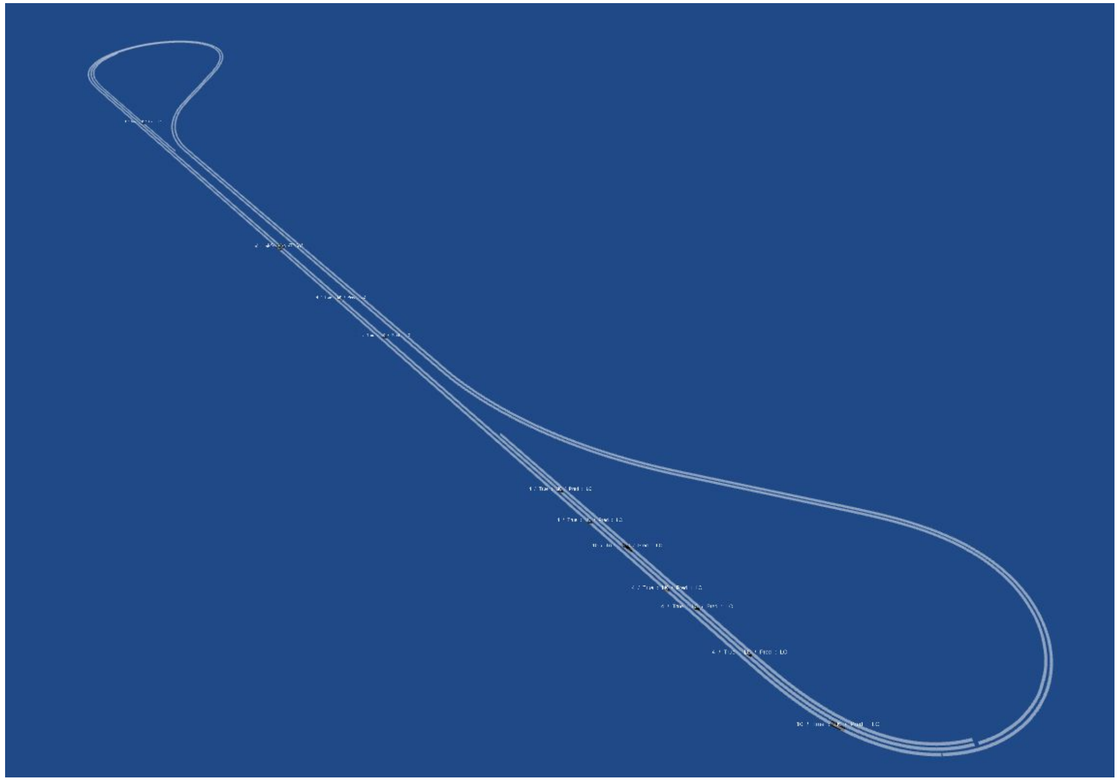
OneDay Project

Project 목표

차량의 차선 변경 Intention 확인

- 도로간 교차가 없는 자동차전용 도로 환경
- 과거 0.5s+현재(11step)의 데이터를 통해서 현재 시점의 **차선 변경 Intention을 확인**





실행 순서

[Catkin_ws]

- src
  - OneDayProject
    - src
      - vehicle_node.py
      - log
        - test 1~6.pickle

[실행 순서]

@ Catkin_ws

- catkin_make
- source devel setup.bash
- roslaunch launch/day.launch

launch > day.launch

```
1  <launch>
2
3      <node pkg="OneDayProject" type="vehicle_node.py" name="vehicle_node" output="screen"/>
4          <param name="map_path" value="$(find OneDayProject)/maps/"/>
5          <param name="SamplePath" value="$(find OneDayProject)/log/" />
6          <param name="SampleId" value= "1" />
7
8
9      <node pkg="rviz" type="rviz" name="my_rviz" args="-d $(find OneDayProject)/simulation.rviz" />
10
11
12     <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py" name="teleop_twist_keyboard" respawn="true"/>
13     <param name="use_sim_time" value="False"/>
14
15
16
17 </launch>
```

Main 실행 파일

수정하며 Test

Keyboard 입력을 받는 pkg

Vehicle node

- Environments 라는 하나의 class 안에서 시뮬레이션이 수행됨.
- 간단한 정보 교환을 위해, 맵이나 차량 데이터는 “self.*” 형태를 가지고 있음.
- 과제의 목표는 “Publish” 함수 내에서 각 차량마다 LC intention을 예측하는 것.

[vehicle_node.py]

Class Environments

-init()

-- init_variable() : 차량 데이터를 Load

-- load_map() : 맵 데이터를 Load

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- loop() : 20Hz 로 loop 반복

-- Publish() : 차량 정보 및 예측 결과를 publish

-- pub_map() : 맵 데이터 publish

- callback_plot() : pause를 위한 subscriber의 callback

코드 구조

Environments Class

```python
class Environments(object):
    def __init__(self):
        rospy.init_node('Environments')

        self.init_variable()
        self.set_subscriber()
        self.set_publisher()
        self.load_map()

        r = rospy.Rate(20)
        while not rospy.is_shutdown():
                            0.05s 기준
            self.loop()
            r.sleep()
```

```python
    def init_variable(self):
        self.pause = False
        self.time = 11
        self.br = tf.TransformBroadcaster()

        SamplePath = rospy.get_param("SamplePath")
        SampleList = sorted(glob.glob(SamplePath+"/*.pickle"))
        SampleId = (int)(rospy.get_param("SampleId"))

        ######################### Load Vehicle ################
        with open(SampleList[SampleId], 'rb') as f:
            self.vehicles = pickle.load(f)
```

[vehicle_node.py]

Class Environments

-init()

-- init_variable() : 차량 데이터를 Load

-- load_map() : 맵 데이터를 Load

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- loop() : 20Hz 로 loop 반복

-- Publish() : 차량 정보 및 예측 결과를 publish

-- pub_map() : 맵 데이터 publish

- callback_plot() : pause를 위한 subscriber의 callback

데이터 구조

차량 데이터

- Agent별로 주행 데이터를 시간에 대한 데이터로 저장

: 각 Agent는 현재 차선 lane_id와 타겟 차선 target_lane_id를 가지고, 전방에 저속 차량이 인지되었을 때 차선 변경을 수행함.

: 총 T step (0.05sec per step) 의 데이터를 저장.

: Id 별로 데이터가 저장(dictionary 형태)

ex) 0번 차량의 데이터 “vehicles[0]”, 데이터 사이즈는 [T*14]

: 14개의 attribute는 다음의 정보를 포함하고 있음.

0 : lane_id

1 : target_lane_id

2 : s (1차선 시작 지점 기준)

3 : d (1차선 기준 감소)

4 : GX (Global X 좌표 (GPS))

5 : GY (Global Y 좌표 (GPS))

6 : Gyaw (Global heading (GPS))

7 : v (차량 속도, m/s)

8 : yawrate (차량 yawrate, rad/s)

9 : Mode (LK, LC mode – GT)

10 : ax (차량 가속도, m^2/s)

11 : steer (차량 steering angle, rad)

12 : length (차량 length)

13 : width (차량 width)

사용 가능한 데이터는 빨간색! 정보
– 관측 가능한 값

코드 구조

Environments Class

```
class Environments(object):
    def __init__(self):
        rospy.init_node('Environments')

        self.init_variable()
        self.set_subscriber()
        self.set_publisher()
        self.load_map()

        r = rospy.Rate(20)
        while not rospy.is_shutdown():

            self.loop()
            r.sleep()
```

[vehicle_node.py]

Class Environments

-init()

-- init_variable() : 차량 데이터를 Load

-- load_map() : 맵 데이터를 Load

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- loop() : 20Hz 로 loop 반복

-- Publish() : 차량 정보 및 예측 결과를 publish

-- pub_map() : 맵 데이터 publish

- callback_plot() : pause를 위한 subscriber의 callback

* Launch한 명령창에서 “u” key를 누르면 정지, “k”를 누르면 재 시작

```
    def set_subscriber(self):
        rospy.Subscriber('/cmd_vel',Twist, self.callback_plot,queue_size=1)

    def set_publisher(self):

        self.sur_pose_plot = rospy.Publisher('/rviz/sur_obj_pose', MarkerArray, queue_size=1)
        self.map_plot = rospy.Publisher('/rviz/maps', MarkerArray, queue_size=1)
        self.text_plot = rospy.Publisher('/rviz/text', MarkerArray, queue_size=1)
```

For Pause

Obj Cube

Map

Obj ID & Result

코드 구조

Environments Class

```python
class Environments(object):
    def __init__(self):
        rospy.init_node('Environments')

        self.init_variable()
        self.set_subscriber()
        self.set_publisher()
        self.load_map()

        r = rospy.Rate(20
        while not rospy.i

            self.loop()
            r.sleep()
```

```python
def load_map(self):
    self.map_file = []
    map_path = rospy.get_param("map_path")
    matfiles = ["waypoints_0_rev.mat",
                "waypoints_1_rev.mat",
                "waypoints_2_rev.mat",
                "waypoints_3_rev.mat"
                ]

    station_offset = [0, 0, 0, 0]

    for i, matfile in enumerate(matfiles):
        mat = sio.loadmat(map_path+matfile)

        easts = mat["east"][0]
        norths = mat["north"][0]
        stations = mat["station"][0]+station_offset[i]
        # [326107, 340838]

        self.map_file.append(np.stack([easts, norths, stations],axis=-1))

    self.D_list = [ 0, -3.85535188 ,-7.52523438, -7.37178602]
    self.D_list = np.array(self.D_list)
```

[vehicle_node.py]

Class Environments

-init()

-- init_variable() : 차량 데이터를 Load

-- load_map() : 맵 데이터를 Load

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- loop() : 20Hz 로 loop 반복

-- Publish() : 차량 정보 및 예측 결과를 publish

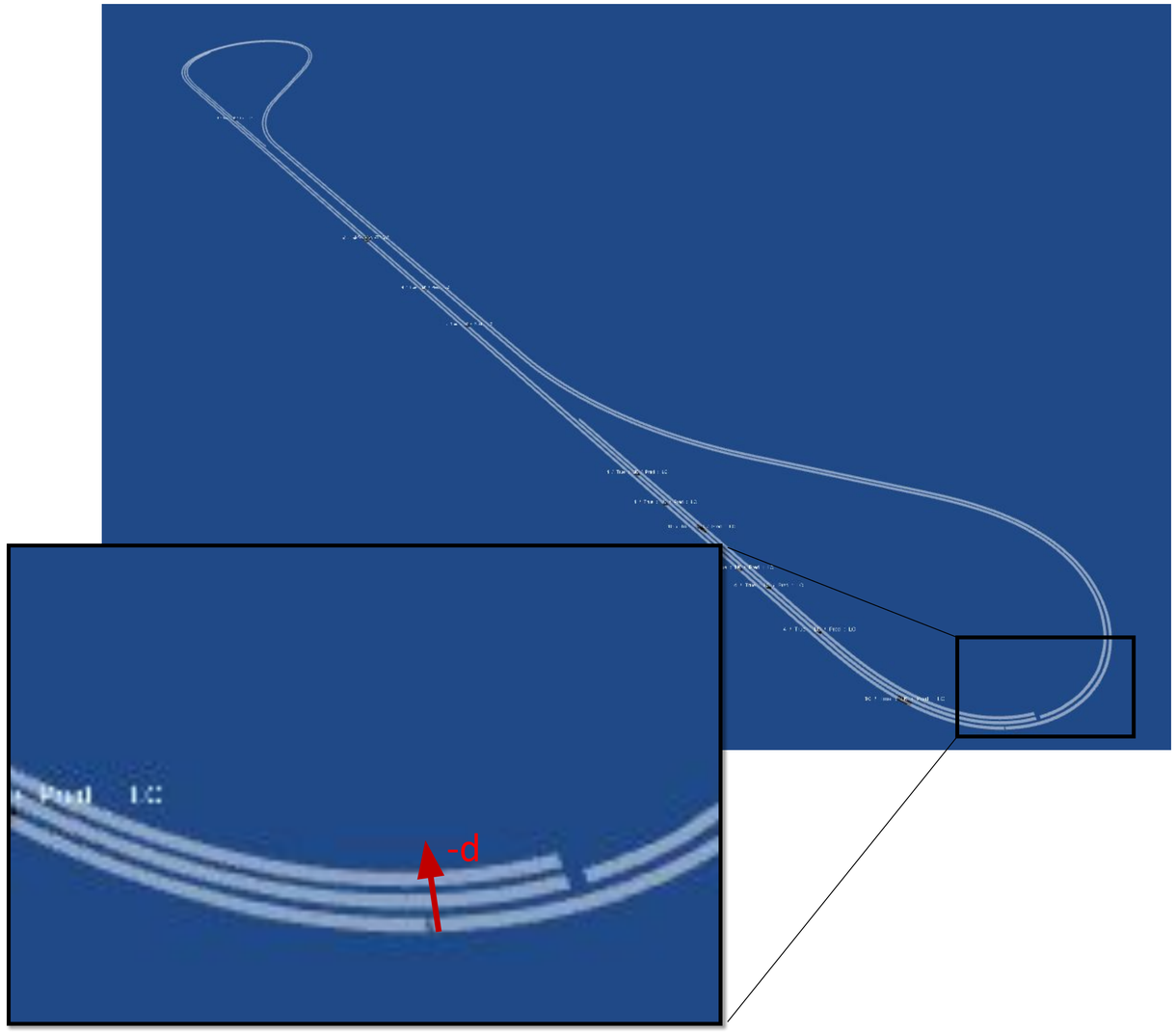
-- pub_map() : 맵 데이터 publish

- callback_plot() : pause를 위한 subscriber의 callback

데이터 구조

맵 데이터

- 총 4개의 차선으로 구성
  : 1차선, 2차선, 합류로, 분기로
- 각 차선별로 [Gx, Gy, station] 값이 저장되어 있음. *Global scale : 326107, 340838
- station은 1차선 시작점 기준으로 작성되어 있으며, d 값은 [0, -3.8553,-7.5234,-7.3717]



OneDay Project

코드 구조

Environments Class

```python
def loop(self):

    if self.pause:
        pass
    else:
        self.publish()
        self.pub_map()
        self.time+=1

    if self.time>=(len(self.vehicles[0])-1):
        rospy.signal_shutdown("End of the logging Time")
```

Publish-1

```python
def publish(self, is_delete = False):

    Objects = MarkerArray()
    Texts = MarkerArray()

    for i in range(len(self.vehicles)):

        ##################################################
        ##################### To Do ######################
        ##################################################
        """
        각 ve[illegible]step(0.5s)의 관측 값을 바탕으로,
        현재 차량의 차선 변경 의도를 예측하기.
        그리고, pred 변수에 LC 이면 "LC", LK이면 "LK" 라는 값을 대입하여 결과를 확인해 봅시다.
        """

        pred = "LC" # or "LK"

        gt = "LC" if self.vehicles[i][self.time][9] == 1 else "LK"
```

self.vehicle[i][self.time-10:self.time+1] 데이터 활용
i 번째 차량마다 결과를 아래 pred에 저장.

Mode

[vehicle_node.py]

Class Environments

-init()

-- init_variable() : 차량 데이터를 Load

-- load_map() : 맵 데이터를 Load

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- loop() : 20Hz 로 loop 반복

-- Publish() : 차량 정보 및 예측 결과를 publish

-- pub_map() : 맵 데이터 publish

- callback_plot() : pause를 위한 subscriber의 callback

* 주의할 점 : d의 경우 1차선 기준으로 되어 있음.
- 차량의 lane_id가 2차선인 경우, d 값에서 2차선에 해당하는 d 값을 빼줘야 2차선과의 distance가 나옴.
  ex) d = self.vehicles[id][time][3] – self.D_list[self.vehicles[id][time][1]]

OneDay Project

코드 구조

Environments Class

```python
def loop(self):

    if self.pause:
        pass
    else:
        self.publish()
        self.pub_map()
        self.time+=1

    if self.time>=(len(self.vehicles[0])-1):
        rospy.signal_shutdown("End of the logging Time")
```

Publish-1

```python
def publish(self, is_delete = False):

    Objects = MarkerArray()
    Texts = MarkerArray()

    for i in range(len(self.vehicles)):

        ##################################################
        ###################### To Do #####################
        ##################################################

        """
        각 [illegible] step(0.5s)의 관측 값을 바탕으로,
        현재 차량의 차선 변경 의도를 예측하기.
        그리고, pred 변수에 LC 이면 "LC", LK이면 "LK" 라는 값을 대입하여 결과를 확인해 봅시다.
        """

        pred = "LC" # or "LK"

        gt = "LC" if self.vehicles[i][self.time][9] == 1 else "LK"
```

self.vehicle[i][self.time-10:self.time+1] 데이터 활용
i 번째 차량마다 결과를 아래 pred에 저장.

Mode

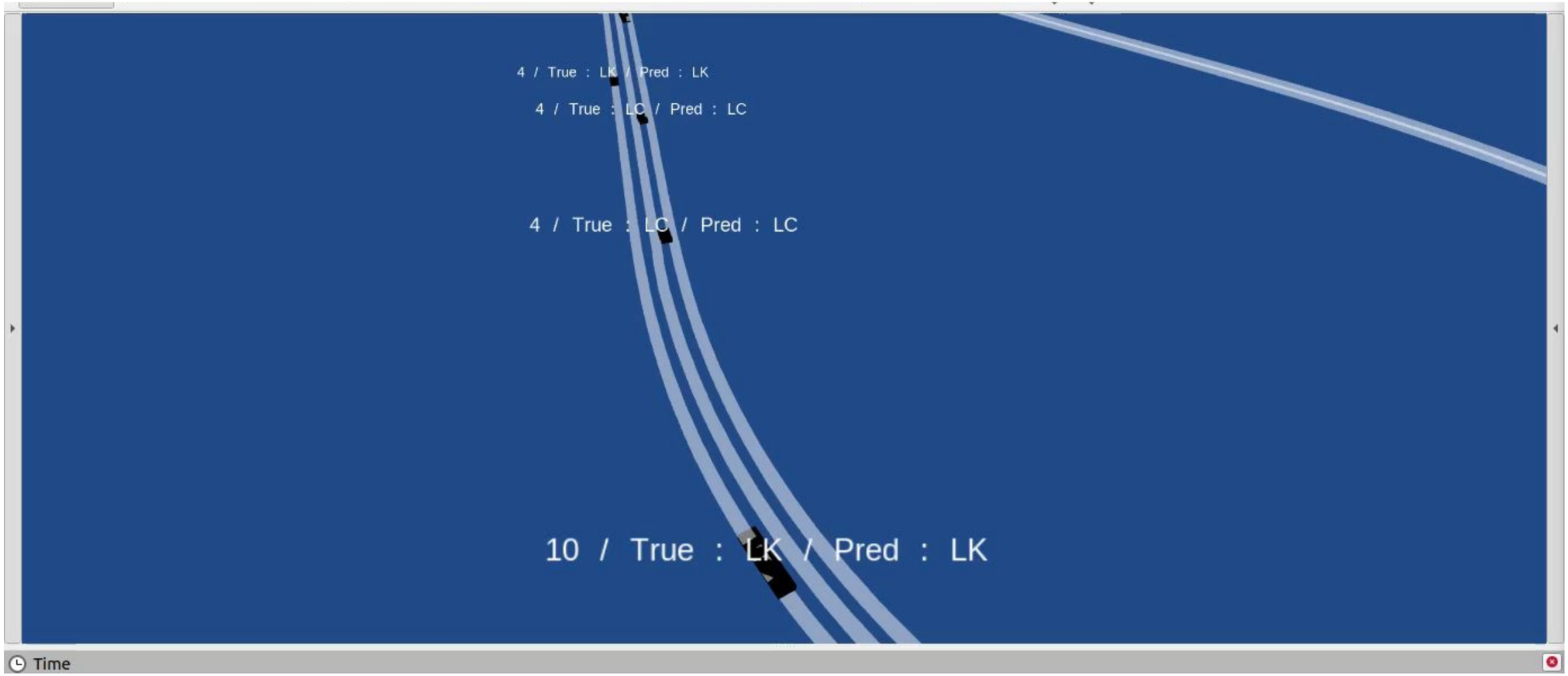
```python
        ## marker_publish
        q = tf.transformations.quaternion_from_euler(0, 0, self.vehicles[i][self.time][6])

        marker = Marker()
        marker.header.frame_id = "world"
        marker.header.stamp = rospy.Time.now()
        marker.id = i
        marker.type = Marker.CUBE

        marker.pose.position.x = self.vehicles[i][self.time][4]
        marker.pose.position.y = self.vehicles[i][self.time][5]
        marker.pose.position.z = 0.5

        marker.pose.orientation.x = q[0]
        marker.pose.orientation.y = q[1]
        marker.pose.orientation.z = q[2]
        marker.pose.orientation.w = q[3]

        marker.scale.x = self.vehicles[i][self.time][12]
        marker.scale.y = self.vehicles[i][self.time][13]
        marker.scale.z = 1
        marker.color.a = 1.0

        Objects.markers.append(marker)

        text = Marker()
        text.header.frame_id = "world"
        text.ns = "text"
        text.id = i
        text.type = Marker.TEXT_VIEW_FACING

        text.action = Marker.ADD

        text.color = ColorRGBA(1, 1, 1, 1)
        text.scale.z = 5
        text.text = str(self.vehicles[i][self.time][12])+" / True : " + gt+" / Pred : "  + pred
        text.pose.position = Point(self.vehicles[i][self.time][4], self.vehicles[i][self.time][5], 3)

        Texts.markers.append(text)

    self.sur_pose_plot.publish(Objects)
    self.text_plot.publish(Texts)

    self.br.sendTransform((self.vehicles[0][self.time][4], self.vehicles[0][self.time][5], 0),
                          tf.transformations.quaternion_from_euler(0, 0,self.vehicles[0][self.time][6]),
                          rospy.Time.now(),
                          "base_link",
                          "world")
```

Publish-2

결과

HMM(Hidden Markov Model)을 활용한 차선 변경 의도 판단

- True Mode를 기준으로 LC / LK 각각에 대한 데이터를 취득 및 학습

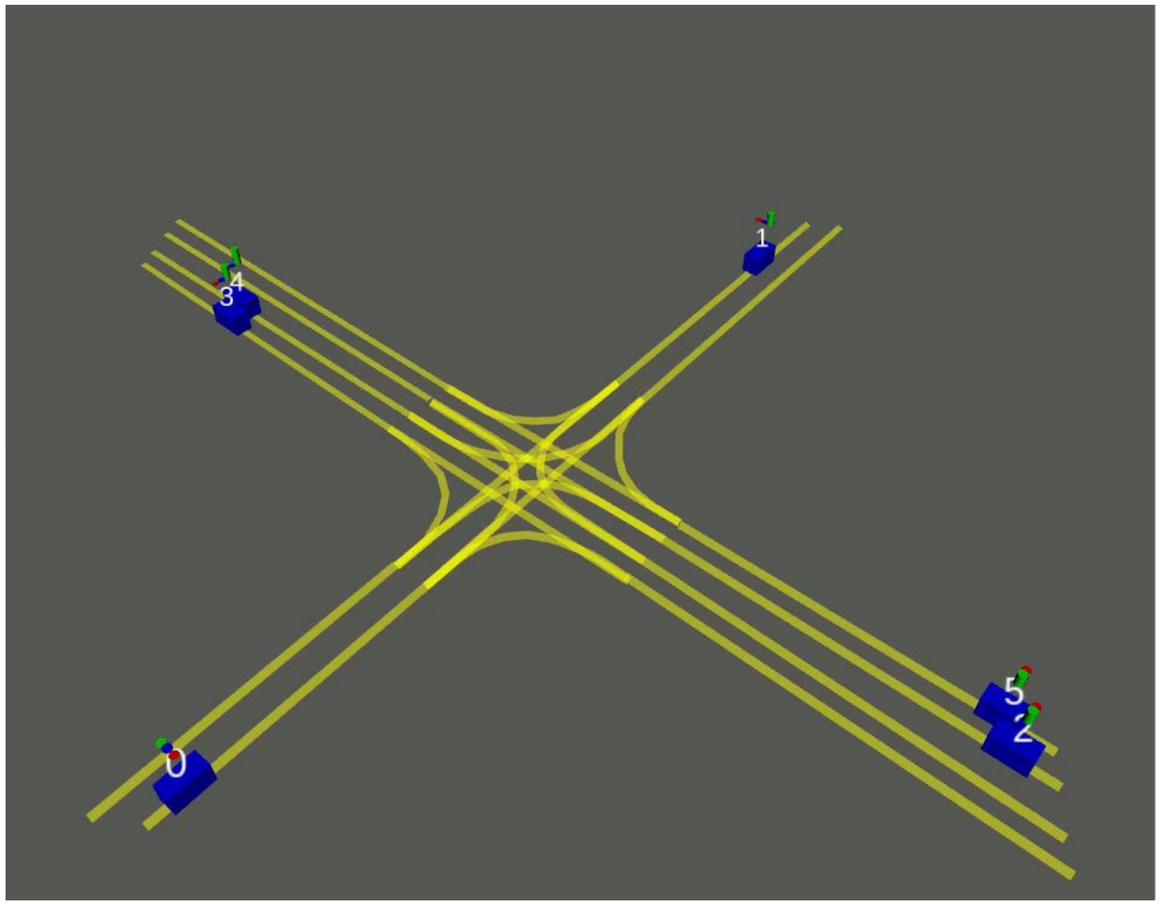


programmers

Week Project

Project 목표

- 도로간 교차가 존재하는 교차로 환경
- 시작 지점에서 교차로를 통과하여 목표 지점 (좌회전, 직진, 우회전) 까지 도달하기가 목표
- 차량에서 주변 agent 정보를 센서로 받는 상황을 가정
  : 항상 relative한 정보를 받게 됨.
- 차선 변경은 없다고 가정.



실행 순서

[Catkin_ws]

- launch
  - simulation.launch
- src
  - vehicle_node
    - src
      - agent.py
      - environment.py
      - main.py
      - utils.py
    - env_info.pickle
  - simulation.rviz

Main 실행 파일

```
<launch>

  <node pkg="vehicle_node" type="main.py" name="vehicle_node" output="screen"/>
      <param name="file_path" value="$(find vehicle_node)/" />


  <node pkg="rviz" type="rviz" name="my_rviz" args="-d $(find vehicle_node)/../../simulation.rviz" />
  <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py" name="teleop_twist_keyboard" respawn="true"/>


  <param name="use_sim_time" value="False"/>

</launch>
```

실행 순서

@ Catkin_ws

- catkin_make
- source devel setup.bash
- roslaunch launch/simulation.launch

Project

Vehicle node

- Main : loop 및 RVIZ visualization 담당.
- Environments class : agents 정보 및 path간 conflict 정보를 통합하여 관리.
- Agent class : agent instance, sensor, controller 등의 정보를 포함하고 있음.

[main.py]

Class Simulation

-init()

-- initialize() : SDV route 정의 / Environments spawn

self.env

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- run() : 10Hz 로 loop 반복

-- **self.env.run()**

-- pub_track() : track 정보를 RVIZ에 publish

-- pub_map() : map 정보를 RVIZ에 publish

-- delete_agent() : 영역을 벗어난 track 정보 삭제

-- self.env.respawn()

- callback_plot() : pause를 위한 subscriber의 callback

[environments.py]

Class Environments

-init()

-- initialize() : map 정보 load / spawn agents

--- spawn agent() : target path sampling (for others)

: target path간 교차점 정보 추출

self.vehicles

Self.int_pt_list : agent spawn

- run() :

-- self.vehicles[id].step_auto():

주변 agent는 auto로 제어

--self.vehicles[0].step_manual():

: SDV의 경우 "ax, steer" 를 입력하여 제어

- delete_agent() : 영역을 벗어난 agent 정보 삭제

- respawn() : 최소 agent 수보다 모자란 경우 respawn

[agents.py]

Class agent

-init() : 현재 속도 및 타겟 속도, 초기 위치 등 sampling

(for others)

- get_local_path() : target path 일부를 local 좌표로 변환

- get_measure() : 주변 agent 일부를 local 좌표로 변환

- step_auto(vehicles, int_pt_list) : 주변 차량 제어

-- lateral_controller() : 차선 유지를 위한 steering angle

-- longitudinal_controller() : 종방향 속도 제어, ax

- step_manual(**ax, steer**) : SDV 제어

코드 구조

Main.py

```python
class Simulation(object):
    def __init__(self, dt=0.1):

        rospy.init_node('simulation')

        self.set_subscriber()
        self.set_publisher()
        self.initialize()

        r = rospy.Rate(10)
        while not rospy.is_shutdown():

            self.run()
            r.sleep()
```

```python
    def set_subscriber(self):
        rospy.Subscriber('/cmd_vel',Twist, self.callback_plot,queue_size=1)


    def set_publisher(self):

        # Object Cube & ID
        self.sur_pose_plot = rospy.Publisher('/rviz/sur_obj_pose', MarkerArray, queue_size=1)
        self.sur_v_plot = rospy.Publisher('/rviz/sur_v_plot', MarkerArray, queue_size=1)
        self.sur_accel_plot = rospy.Publisher('/rviz/sur_accel_plot', MarkerArray, queue_size=1)
        self.sur_decel_plot = rospy.Publisher('/rviz/sur_decel_plot', MarkerArray, queue_size=1)

        self.text_plot = rospy.Publisher('/rviz/text', MarkerArray, queue_size=1)

        # Map Point
        self.map_plot = rospy.Publisher('/rviz/maps', MarkerArray, queue_size=1)
```

[main.py]

Class Simulation

-init()

-- initialize() : SDV route 정의 / Environments spawn

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- run() : 10Hz 로 loop 반복

-- self.env.run()

-- pub_track() : track 정보를 RVIZ에 publish

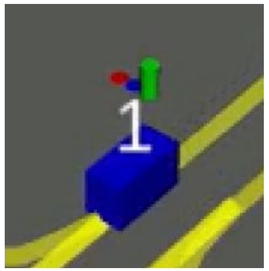
-- pub_map() : map 정보를 RVIZ에 publish

-- delete_agent() : 영역을 벗어난 track 정보 삭제

-- self.env.respawn()

- callback_plot() : pause를 위한 subscriber의 callback

decel accel vel



Week Project

코드 구조

Main.py

```python
class Simulation(object):
    def __init__(self, dt=0.1):

        rospy.init_node('simulation')

        self.set_subscriber()
        self.set_publisher()
        self.initialize()

        r = rospy.Rate(10)
        while not rospy.is_shutdown():

            self.run()
            r.sleep()
```

```python
    def initialize(self):
        self.pause = False
        course_idx = 0 # 0 : 좌회전 / 1 : 직진 / 2 : 우회전
        self.env = Environments(course_idx)

    def run(self):

        if self.pause:
            pass
        else:
            self.env.run()
            self.pub_track()
            self.pub_map()

            self.delete_agent()
            self.env.respawn()
```

Self.env에 Environment 저장

Self.env에서 run 수행

[main.py]

Class Simulation

-init()

-- initialize() : SDV route 정의 / Environments spawn

-- set_publisher() : RVIZ plot을 위한 publisher 정의

-- set_subscriber() : pause를 위한 subscriber 정의

- run() : 10Hz 로 loop 반복

-- self.env.run()

-- pub_track() : track 정보를 RVIZ에 publish

-- pub_map() : map 정보를 RVIZ에 publish

-- delete_agent() : 영역을 벗어난 track 정보 삭제

-- self.env.respawn()

- callback_plot() : pause를 위한 subscriber의 callback

코드 구조

Environments.py

```python
class Environments(object):
    def __init__(self, course_idx, dt=0.1, min_num_agent=8):

        self.spawn_id = 0
        self.vehicles = {}
        self.int_pt_list = {}
        self.min_num_agent = min_num_agent
        self.dt = dt
        self.course_idx = course_idx

        self.initialize()

    def initialize(self, init_num=6):

        self.pause = False
        filepath = rospy.get_param("file_path")
        Filelist = glob.glob(filepath+"/*info.pickle")

        file = Filelist[0]

        with open(file, "rb") as f:
            Data = pickle.load(f)

        self.map_pt = Data["Map"]
        self.connectivity = Data["AS"]

        for i in range(init_num):
            if i==0:
                CourseList = [[4,1,18], [4,2,25], [4,0,11]]
                self.spawn_agent(target_path = CourseList[self.course_idx], init_v = 0)
            else:
                self.spawn_agent()
```

Map 정보 load

[environments.py]

Class Environments

-init()

-- initialize() : map 정보 load / spawn agents

--- spawn agent() : target path sampling (for others)

: target path간 교차점 정보 추출

: agent spawn

- run() :

-- self.vehicles[id].step_auto():

주변 agent는 auto로 제어

--self.vehicles[0].step_manual():

: SDV의 경우 “ax, steer” 를 입력하여 제어

- delete_agent() : 영역을 벗어난 agent 정보 삭제

- respawn() : 최소 agent 수보다 모자란 경우 respawn

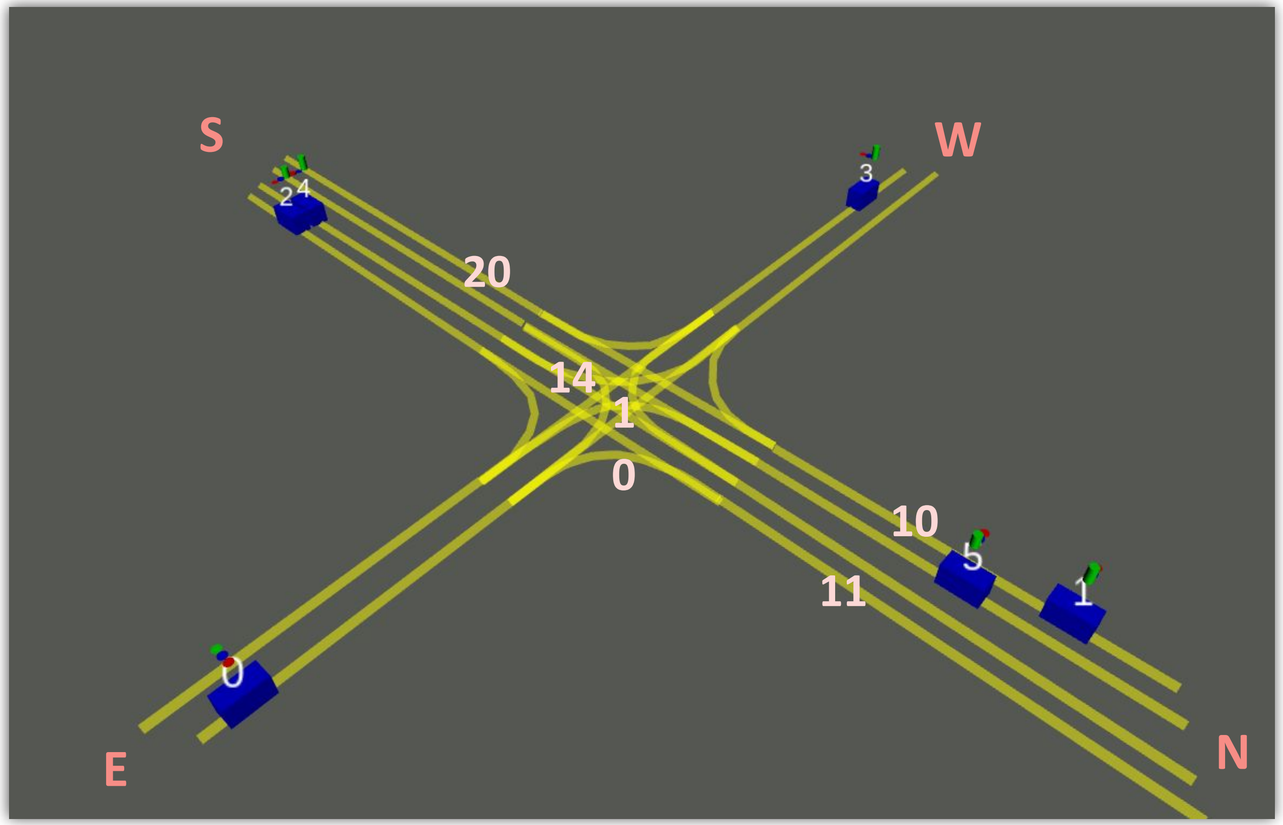
Week Project

코드 구조

맵 데이터

- data[“Map”] : MapPoint
  > 26개의 차선, dictionary 형태로 되어 있음.
  > 각 차선마다 **[X, Y, Heading, R]** list가 저장되어 있음.
- data[“AS”] : 차선 간 연결 정보
  > 행 index에 해당하는 차선과 연결되는 다음 차선의 정보(열 index)
  ex) data[“Connectivity”][0,11] = 1

<Index>
0: E2N
1: E2S
2: E2W
3: E2E
4: E2W
5: N2E
6: N2S(1)
7: N2S(2)
8: N2W
9: N2N(1)
10: N2S(1)
11: N2N(2)
12: N2S(2)
13: S2E
14: S2N(1)
15: S2N(2)
16: S2W
17: S2N(1)
18: S2S(1)
19: S2N(2)
20: S2S(2)
21: W2E
22: W2N
23: W2S
24: W2E
25:W2W



코드 구조

Environments.py

```python
class Environments(object):
    def __init__(self, course_idx, dt=0.1, min_num_agent=8):

        self.spawn_id = 0
        self.vehicles = {}
        self.int_pt_list = {}
        self.min_num_agent = min_num_agent
        self.dt = dt
        self.course_idx = course_idx

        self.initialize()

    def initialize(self, init_num=6):

        self.pause = False
        filepath = rospy.get_param("file_path")
        Filelist = glob.glob(filepath+"/*info.pickle")

        file = Filelist[0]

        with open(file, "rb") as f:
            Data = pickle.load(f)

        self.map_pt = Data["Map"]
        self.connectivity = Data["AS"]

        for i in range(init_num):
            if i==0:
                CourseList = [[4,1,18], [4,2,25], [4,0,11]]
                self.spawn_agent(target_path = CourseList[self.course_idx], init_v = 0)
            else:
                self.spawn_agent()
```

초기 설정 수만큼 agent spawn,
SDV는 id=0, init_v = 0, target_path를 정의해주고 spawn

[environments.py]

Class Environments

-init()

-- initialize() : map 정보 load / spawn agents

--- spawn agent() : target path sampling (for others)

: target path간 교차점 정보 추출

: agent spawn

- run() :

-- self.vehicles[id].step_auto():

주변 agent는 auto로 제어

--self.vehicles[0].step_manual():

: SDV의 경우 “ax, steer” 를 입력하여 제어

- delete_agent() : 영역을 벗어난 agent 정보 삭제

- respawn() : 최소 agent 수보다 모자란 경우 respawn

Week Project

코드 구조

Spawn agent

- SDV
  - 특정 차선(4) 에 속도가 0인 상태로 spawn됨.

- 주변 agent
  - Random으로 spawn 됨. SDV의 시작 lane에는 spawn되지 않음.

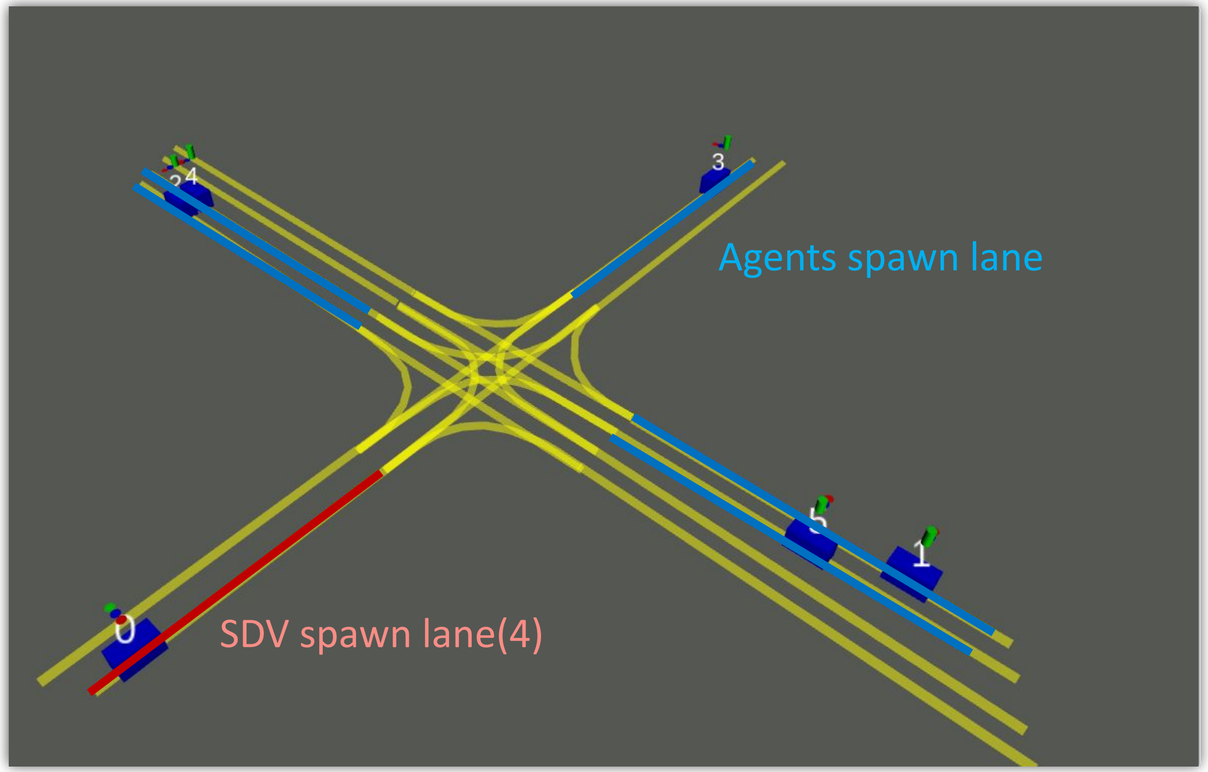
[Spawn Process]
1) target_path가 있는 경우 (SDV)
   (1) initial vel = 5, initial s = 5로 spawn

2) target_path가 없는 경우
   (1) 시작 lane을 random choice
   (2) 급감속을 유발할 수 있는 agent확인
   (3) spawn

Spawn_agent 1

```python
def spawn_agent(self, target_path=[], init_v = None):

    is_occupied = True

    if target_path:

        spawn_cand_lane = target_path[0]
        is_occupied = False
        s_st = 5

    else:
        spawn_cand_lane = [10,12,24,17,19]

        s_st = np.random.randint(0,20)
        max_cnt = 10
        while(is_occupied and max_cnt>0):

            spawn_lane = np.random.choice(spawn_cand_lane)

            is_occupied = False
            for id_ in self.vehicles.keys():
                if (self.vehicles[id_].lane_st == spawn_lane) and np.abs(self.vehicles[id_].s - s_st) < 25:
                    is_occupied = True

            max_cnt-=1
```

For SDV



Week Project

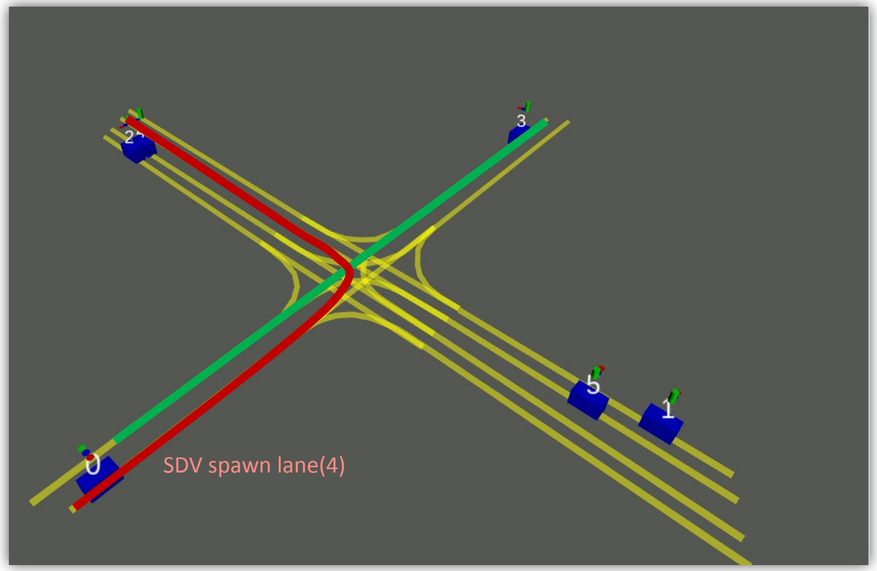
코드 구조

Spawn agent

Spawn_agent 2

```
if is_occupied is False:
    if target_path:
        target_path = target_path

    else:
        target_path = [spawn_lane]
        spawn_lane_cand = np.where(self.connectivity[spawn_lane]==1)[0]

        while(len(spawn_lane_cand)>0):
            spawn_lane = np.random.choice(spawn_lane_cand)
            target_path.append(spawn_lane)
            spawn_lane_cand = np.where(self.connectivity[spawn_lane]==1)[0]

    target_pt = np.concatenate([self.map_pt[lane_id][:-1,:] for lane_id in target_path], axis=0)
    self.int_pt_list[self.spawn_id] = {}

    for key in self.vehicles.keys():
        intersections = find_intersections(target_pt[:,:3], self.vehicles[key].target_pt[:,:3]) # ((x,y), i, j)

        if intersections:
            self.int_pt_list[self.spawn_id][key] = [(inter, xy[0], xy[1]) for (inter, xy) in intersections]
            self.int_pt_list[key][self.spawn_id] = [(inter, xy[1], xy[0]) for (inter, xy) in intersections]

    stopline_idx = len(self.map_pt[target_path[0]])-1
    endline_idx = len(self.map_pt[target_path[0]])+len(self.map_pt[target_path[1]])-2

    self.vehicles[self.spawn_id] = agent(self.spawn_id, target_path, s_st, target_pt, dt=self.dt, init_v = init_v,
                                         stoplineidx = stopline_idx, endlineidx = endline_idx)
    self.spawn_id +=1
```

초기 lane으로부터 connectivity 정보를 통해 target path 선정



이미 Spawn된 agent의 target path와 교차점 정보 저장
Int_pt_list[0,3] = [(x,y), inter_idx_0, inter_idx_3]
Int_pt_list[3,0] = [(x,y), inter_idx_3, inter_idx_0]

Agent_spawn, self.vehicles에 정보가 담김.

코드 구조

Environments.py

```
132     def run(self):
133
134         for id_ in self.vehicles.keys():
135             if id_ == 0:
136                 sensor_info = self.vehicles[id_].get_measure(self.vehicles)
137                 local_lane_info = self.vehicels[id_].get_local_path()
138                 """
139                 To Do
140
141                 """
142
143                 self.vehicles[id_].step_manual(ax = 0.0, steer = 0)
144
145
146
147             if id_  > 0 :
148                 self.vehicles[id_].step_auto(self.vehicles, self.int_pt_list[id_])
149
```

* Sensor_info : 전방 [-60,60] 사이 object의 relative information (List)
  - [obj id, rel x, rel y, rel h, rel vx, rel vy]
  - noise가 섞인 데이터

* local_lane_info : 추종해야 할 lane info [x, y, h, R], local 좌표계

[environments.py]

Class Environments

-init()

-- initialize() : map 정보 load / spawn agents

--- spawn agent() : target path sampling (for others)

: target path간 교차점 정보 추출

: agent spawn

- run() :

-- self.vehicles[id].step_auto():

주변 agent는 auto로 제어

--self.vehicles[0].step_manual():

: SDV의 경우 "ax, steer" 를 입력하여 제어

- delete_agent() : 영역을 벗어난 agent 정보 삭제

- respawn() : 최소 agent 수보다 모자란 경우 respawn

[To Do]

아래의 정보들을 활용하여, SDV가 주변 agent와 충돌 없이 교차로를 통과하여 target lane에 가기 위한 종 / 횡 방향 제어기 설계

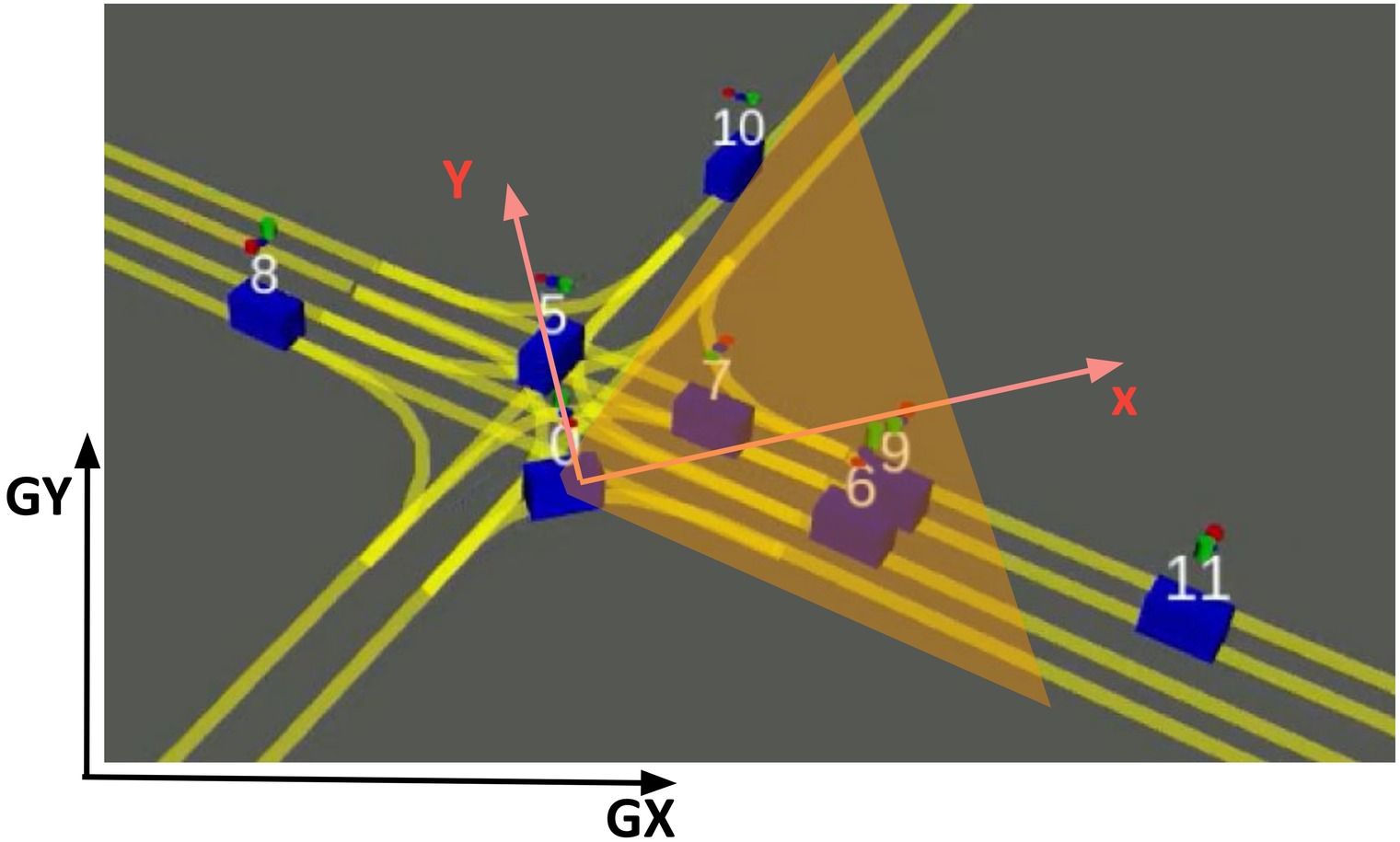
- sensor info
- local lane info
- SDV info : self.vehicles[id_].~ [x, y, h, v, s, d]
- Global map info : self.map_pt / self.connectivity\

다른 agent의 정보에도 접근 가능하지만!!!!
접근을 하지 않는 걸로 먼저 시도해보기!

코드 구조

Environments.py

Sensor 좌표계



- Self.vehicles에서 관리되는 정보는 global 좌표계

  ex) agent 0 (SDV) : x = 5, y = 3, h = 30(deg), v = 10

  agent 10 : x = 7, y = 8, h = -150(deg), v = 10

- get_measure() 로 받는 정보는 local 좌표계

  ex) agent 10 : x = 3, y = 5, h -120(deg), vx = -5 , vy = -7

코드 구조

Agent.py

간단한 kinematic model 사용하여 차량 움직임 구현

```python
def step_auto(self, vehicles, int_pt_list):

    self.steer = self.lateral_controller()
    ax = self.longitudianal_controller(vehicles, int_pt_list)

    dax = np.clip(ax-self.ax, -1,1)
    self.ax +=dax
    self.ax = np.clip(self.ax, -self.v / self.dt, 5)

    self.v += self.ax*self.dt

    if self.v < 1:
        self.steer = np.clip(self.steer, -10/57.3, 10/57.3)

    self.h += self.steer*self.dt
    self.x += np.cos(self.h)*self.v*self.dt
    self.y += np.sin(self.h)*self.v*self.dt

    self.s, self.d = get_frenet(self.x, self.y, self.target_pt[:,0], self.target_pt[:,1], self.target_s)
    self.s_idx = bisect.bisect_left(self.target_s, self.s)
```

```python
def step_manual(self, ax, steer):

    ax = np.clip(ax, -self.v / self.dt, 5)
    self.v += ax*self.dt

    self.h += steer*self.dt
    self.x += np.cos(self.h)*self.v*self.dt
    self.y += np.sin(self.h)*self.v*self.dt

    self.s, self.d = get_frenet(self.x, self.y, self.target_pt[:,0], self.target_pt[:,1], self.target_s)
    self.s_idx = bisect.bisect_left(self.target_s, self.s)
```

주변 agent의 경우에는 서로 간의 정보를 알고 있다는 가정 하에 자동으로 제어됨.
(env.vehicles와 env.int_pt_list에 접근 가능)

코드 구조

Agent.py

Lateral controller – Pure Pursuit 제어

```python
def step_auto(self, vehicles, int_pt_list):

    self.steer = self.lateral_controller()
    ax = self.longitudianal_controller(vehicles, int_pt_list)

    dax = np.clip(ax-self.ax, -1,1)
    self.ax +=dax
    self.ax = np.clip(self.ax, -self.v / self.dt, 5)

    self.v += self.ax*self.dt

    if self.v < 1:
        self.steer = np.clip(self.steer, -10/57.3, 10/57.3)

    self.h += self.steer*self.dt
    self.x += np.cos(self.h)*self.v*self.dt
    self.y += np.sin(self.h)*self.v*self.dt

    self.s, self.d = get_frenet(self.x, self.y, self.target_pt[:,0], self.target_pt[:,1], self.target_s)
    self.s_idx = bisect.bisect_left(self.target_s, self.s)
```

```python
def lateral_controller(self):

    path = self.get_local_path()
    self.local_path = path

    if len(path)==0:
        delta = 0

    else:
        lookahead_dist = self.v*1

        target_idx = np.where(path[:,0]>=lookahead_dist)[0]

        if len(target_idx)==0:
            target_idx = len(path)-1
        else:
            target_idx = target_idx[0]

        target_x = path[target_idx, 0]
        target_y = path[target_idx, 1]

        r = (target_x**2+target_y**2)/(2*target_y+1e-2)

        delta = np.arctan2(6/r, 1)

    return delta
```

코드 구조

Agent.py

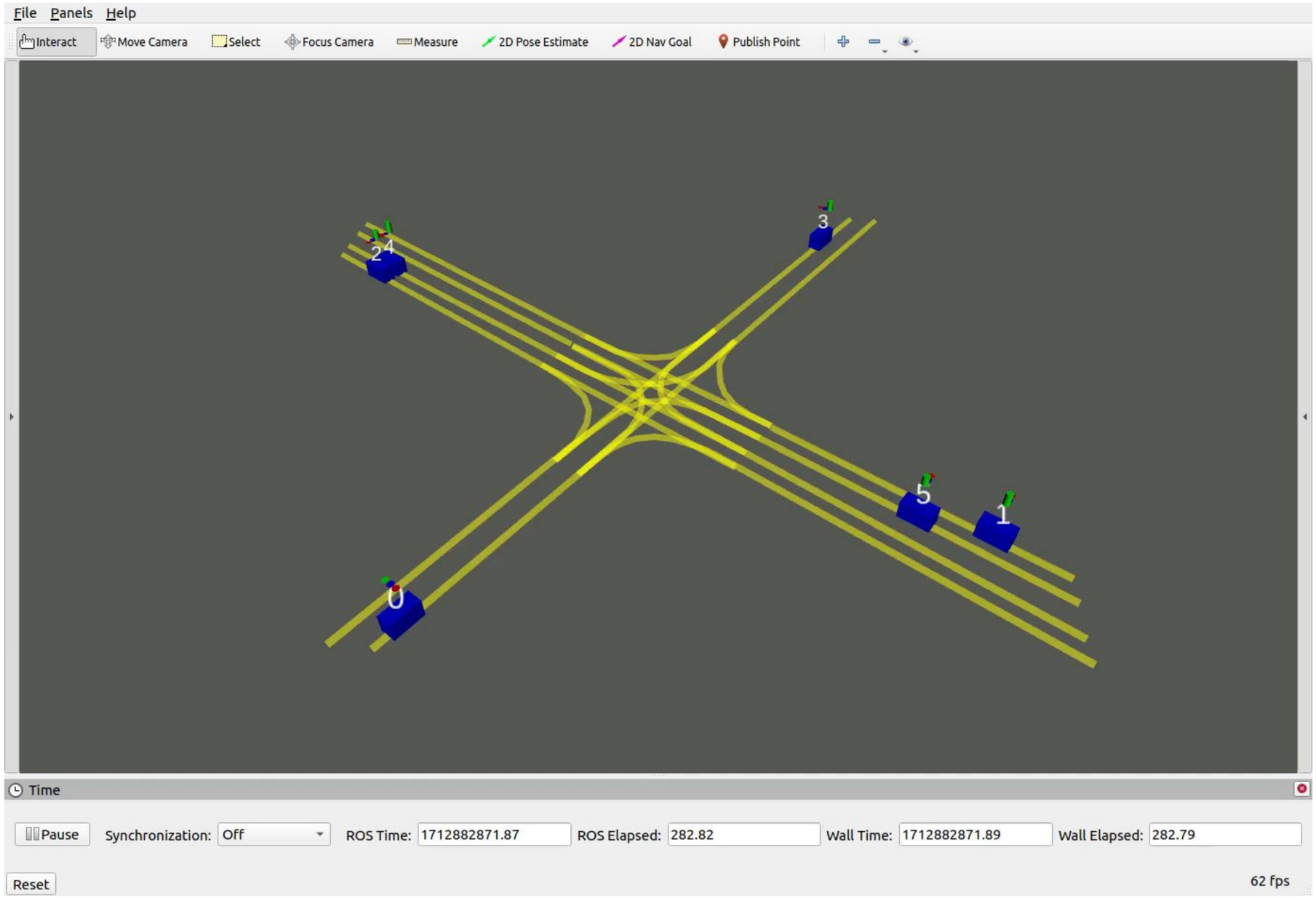
Longitudinal Controller – 관심 있는 차량에 대해 충돌을 피하기 위한 감가속도 / 타겟 속도 추종을 위한 감가속도 / curve를 돌기 위한 감가속도 비교

```python
def step_auto(self, vehicles, int_pt_list):

    self.steer = self.lateral_controller()
    ax = self.longitudianal_controller(vehicles, int_pt_list)

    dax = np.clip(ax-self.ax, -1,1)
    self.ax +=dax
    self.ax = np.clip(self.ax, -self.v / self.dt, 5)

    self.v += self.ax*self.dt

    if self.v < 1:
        self.steer = np.clip(self.steer, -10/57.3, 10/57.3)

    self.h += self.steer*self.dt
    self.x += np.cos(self.h)*self.v*self.dt
    self.y += np.sin(self.h)*self.v*self.dt

    self.s, self.d = get_frenet(self.x, self.y, self.target_pt[:,0], self.target_pt[:,1], self.target_s)
    self.s_idx = bisect.bisect_left(self.target_s, self.s)
```

주변 agent의 경우에는 서로 간의 정보를 알고 있다는 가정 하에 자동으로 제어됨.
(env.vehicles와 env.int_pt_list에 접근 가능)

```python
def longitudianal_controller(self, vehicles, int_pt_list):

    ax_list = self.get_a_agent(vehicles, int_pt_list)

    # add non-front vehicle case
    ax_tarv = self.IDM((1e3, 0, -1e2))
    ax_list.append(ax_tarv)

    # add curvature case
    ax_curv = self.get_a_curvature()
    ax_list.append(ax_curv)

    min_idx = np.argmin(np.array(ax_list))
    ax = ax_list[min_idx]

    return np.clip(ax, -5,5)
```

Week Project



Information

[좌표계 통일]

- SDV에서 “sensor info”를 통해 받는 정보는 relative information.
- 즉, object가 같은 위치에 있어도 SDV의 위치가 바뀌면 다른 정보가 들어옴. History 정보를 쓰고 싶은 경우 좌표계 통일이 유리.

[맵 정보 활용]

- 차선 간의 Conflict position 정보를 미리 저장해 놓고 활용 가능.
- Connectivity 정보를 바탕으로 다음에 갈 수 있는 경로 후보들을 예측해볼 수 있음.

[주변 agent 예측]

- 경로를 예측하셔도 되고,
- 멈출지 말지 의도를 예측하셔도 되고,
- 예측 없이 일정 범위 안에 없으면 통과하셔도 됩니다.

Information

1) Sensor 정보 수신

2) 자 차량 정보를 활용하여 global 좌표로 변환

3) 노이즈 필터링

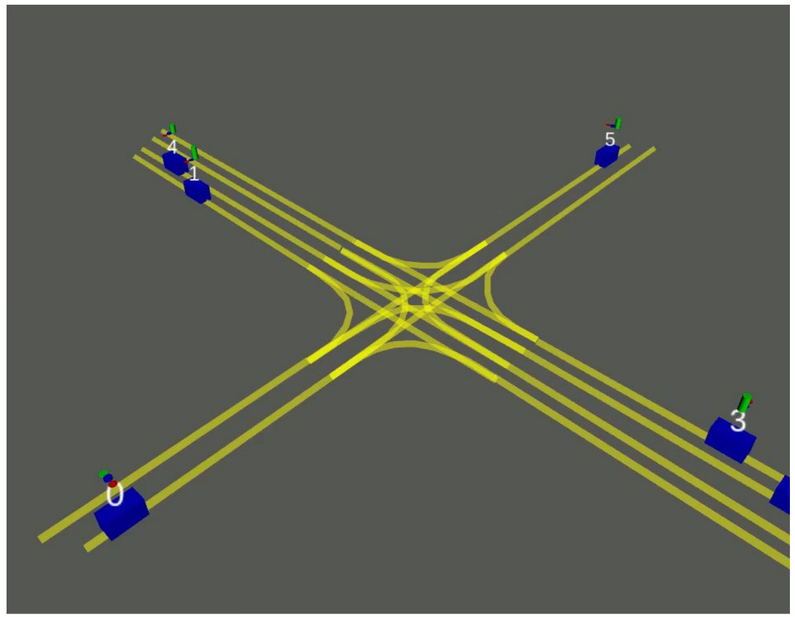
4) Agent별 현재 차선 후보 탐색

5) Agent별 갈 수 있는 차선 후보 탐색

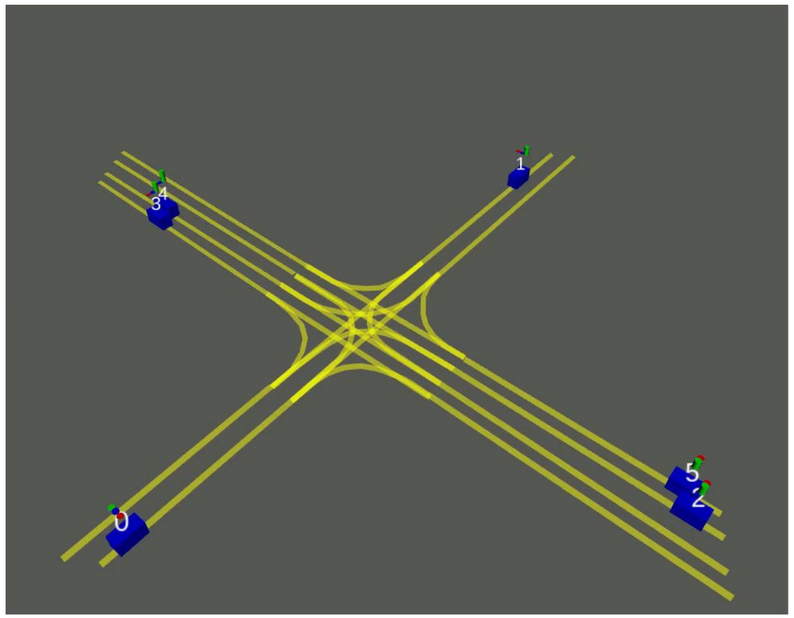
6) Agent별 경로 혹은 의도 예측

7) 주변 agent에 따른 SDV 종 / 횡 방향 제어

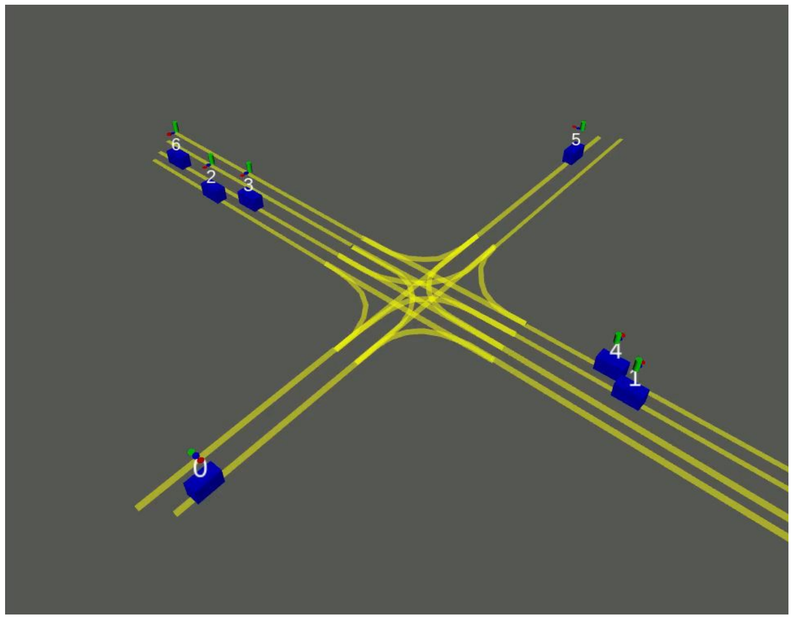
Results



[좌회전]



[직진]



[우회전]