프로젝트 보고서

〈 AEB, Wall Follower 〉

프로젝트	2021학년도 동계 학부연구생
프로젝트 기간	2021.12.20 ~ 2022.01.07
이 름	신건희

1. Automatic Emergency Braking

Yotube Link: https://youtu.be/MwRyheFMkqw

1. Abstract

2021학년도 2학기 학부연구생 프로그램에서 F1TENTH 플랫폼을 제작했다. 이어 동계 방학에는 Pure Pursuit, Scan Matching 등 다양한 자율주행 기술을 F1TENTH 플랫폼에 적용하고 학습하는 과정에 있다. 그리고 마지막에는 앞의 과정을 통해 배운 지식을 바탕으로 자율주행 시나리오를 만들고 직접 구현할 계획이다. 22.01.09 기준으로 Automatic Emergency Braking, Wall Following 기술을 F1TENTH simulator에 적용했고 실제 플랫폼에 옮기는 과정에 있다.

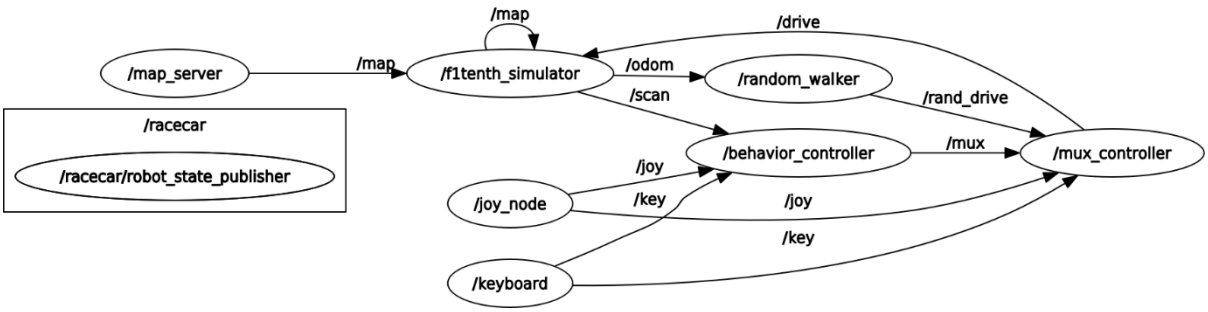
2. Concept

1. TTC(Time to Collision)

- TTC=$\frac{r}{[-\dot{r}]_+}$, r: vehicle과 장애물 사이의 거리, $\dot{r}$: 시간에 따른 r의 변화
- vehicle이 현재 속도와 방향을 유지했을 때 장애물과 부딪히는 시간이다.
- Simulator의 Lidar는 -270~270도의 스캔 범위를 일정 각도를 증가시키면서 1080번 Laser를 송수신한다. 그러므로 TTC는 매 송수신마다 반복되어야 한다.

2. F1TENTH Simulator

- 실제 F1TENTH Race Car에 적용하기전 이상 유무를 확인하기 위한 Simulator이다.
- /joy_node, /keyboard에서 /behavior controller 노드로 향하는 Topic 은 Bool의 형태로 Default는 False이다. 해당 노드를 실행하면 True로 변하며 /mux_cotroller가 해당 노드에서 Publish한 Topic에 따라 Vehicle을 구동하게 한다.
- /joy_node, /keyboard에서 /mux_cotroller 노드로 향하는 Topic은 Vehicle의 Pose를 결정하는 변수들이 포함되어 있다.



<FIG.1 Simulator initial rqt_graph>

3. Safety_node

- AEB 를 구현하기 위해서는 새로운 노드를 만들어야 하고 본 프로젝트에서는 "Safety_node"라 했다.
- 먼저 해당 노드는 f1tenth_simulator 노드로부터 scan Topic 을 통해 LaserScan 데이터를 받고 odom Topic 을 통해 Odometry 데이터를 이용하여 TTC 를 계산한다.
- 만약 TTC 가 설정한 Threshold 보다 작으면 AEB 를 작동한다. 먼저 /behavior 노드에 /brake_bool Topic 을 통해 True 를 Publish 하여 /mux_controller 노드에서 해당 노드에서 Publish 한 /brake Topic 을 Subscribe 하도록 한다.
- 해당 노드에서 /mux_controller 노드에 Publish 한 brake Topic 은 Ackermann Type 으로 차량의 속도를 0 으로 제한하는 Message 다.

```
gunhee@gunhee-950XCJ-951XCJ-950XCR:~/catkin_ws$ rosmsg show nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

```
gunhee@gunhee-950XCJ-951XCJ-950XCR:~/catkin_ws$ rosmsg show ackermann_msgs/AckermannDriveStamped

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
ackermann_msgs/AckermannDrive drive
  float32 steering_angle
  float32 steering_angle_velocity
  float32 speed
  float32 acceleration
  float32 jerk
```

3. Algorithm

1. def__init__(self)

- Node를 실행할 때 Subscribe할 토픽의 이름, Type, Callback 함수를 설정하고 Publish할 토픽의 이름, Type, queue_size를 설정한다.
- 해당 노드에서는 f1tenth_simulator에서 odom, scan 토픽을 Subscribe하여 TTC를 계산

한다.

- 또한, TTC가 설정한 Threshold보다 작을 때 Vehicle를 정지하기 위해 brake_bool, brake 토픽을 Publish한다.

- Threshold는 Trial and Error를 통해 0.3으로 설정했다.

```
gunhee@gunhee-950XCJ-951XCJ-950XCR:~/catkin_ws$ rosmsg show nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

<FIG.2 Odometry Message Type>

```
gunhee@gunhee-950XCJ-951XCJ-950XCR:~/catkin_ws$ rosmsg show sensor_msgs/LaserScan
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

<FIG.3 LaserScan Message Type>

```
gunhee@gunhee-950XCJ-951XCJ-950XCR:~/catkin_ws$ rosmsg show ackermann_msgs/AckermannDriveStamped

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
ackermann_msgs/AckermannDrive drive
  float32 steering_angle
  float32 steering_angle_velocity
  float32 speed
  float32 acceleration
  float32 jerk
```

<FIG.4 ackermannDrivedStamped Message Type>

==========

```
def __init__(self):

        # node subscribes to the odom topic which is of type nav_msgs/Odometry.
        rospy.Subscriber("odom", Odometry, self.odom_callback)
    # node subscribes to the scan topic which is of type sensor_msgs/LaserScam.
        rospy.Subscriber("scan", LaserScan, self.scan_callback)
        #receive 'scan' topic

       # node is publishing to the brake_bool topic using the message type Bool
        self.pub_brake_bool= rospy.Publisher("brake_bool", Bool, queue_size=10)
        # node is publishing to the /brake topic using the message type
AckermannDrivestamped
        self.pub_brake = rospy.Publisher("/brake", AckermannDriveStamped,
queue_size=10)

        self.threshold = 0.3
        self.breakspeed = 0
```

==========

2. def scan_callback(self, scan_msg)

- TTC를 계산하기 위해 Subscribe한 odom, scan Topic으로부터 필요한 변수들을 설정한다. scan 토픽에서 angle_min, angle_increment, ranges를 odom Topic에서는 velocity를 사용한다.
- TTC가 Threshold보다 작을 때 Vehicle를 정지하기 위해 보낼 메시지를 생성한다. Velocity 변수가 포함된 AckermannDriveStamped() Type과 Bool 변수가 포함된 Bool() Type 메시지를 생성한다.
- 모든 Laser Point에 대해 TTC를 계산해야 하므로 for문을 이용한다. Angle_min으로부터 각각의 Laser Point의 각도는 theta로 표현했다.
- 전방에 있는 장애물에 대해서만 AEB가 작동하기 위해서 FOV를 조정한다. 본 프로젝트에서는 Vehicle의 전방에서 좌우로 18도로 설정했다. 즉, 고려할 theta는 -pi/10~pi/10이다
- TTC= $\frac{r}{[-\dot{r}]_+}$ 에서 r은 ranges[i]로 설정했고 $[-r]_+$ 는 Vehicle의 속도 벡터를 Laser Point로 내적 한 것으로 설정했다.
- TTC가 Threshold보다 작다면 True로 설정한 Bool Type의 메세지를 behavior_controller 노드에 Publish하고 Velocity=0으로 설정한 AckermannDriveStamped() Type의 메시지를 mux_controller 노드에 Publish한다.

==========

```
def scan_callback(self, scan_msg):

        self.scan=scan_msg   # receive scan_msg
```

```
        angle_min=scan_msg.angle_min
        # Lidar minumum angle: -2.355rad
        angle_increment = scan_msg.angle_increment
        # Laser angle increment: 4.71/1080[rad], scan beams:1080
        ranges = scan_msg.ranges
        # Distance between vehicle and obstacle
        velocity = self.odom.twist.twist.linear.x   # vehicle velocity

        brake_bool_msg=Bool()                       # message which type is Bool
        brake_bool_msg.data=False
        # update parameter 'data' in brake_bool_msg as False

        brake_msg=AckermannDriveStamped()           # message which type is Bool
        brake_msg.drive.speed=self.breakspeed
        # update parameter 'data' in brake_bool_msg as False


        for i in range(len(ranges)):                # for every laser point
            theta = angle_min + angle_increment * i
        # calculate theta based on angle_min(-2.355rad)

            if velocity != 0:
                velocity = abs(velocity)
        # to be robust from noise(negative)
                if -pi/10 < theta < pi/10:
        # set the FOV(Field of View)
                    TTC = ranges[i] / (velocity*cos(theta))
        # Cacluate TTC
                    if TTC < self.threshold:
        # if TTC is smaller than threshold, then vehicle must be stopped
                        brake_bool_msg.data=True
        # update parameter 'data' in brake_bool_msg as False
                        self.pub_brake_bool.publish(brake_bool_msg)
        # publish brake_bool_msg
                        self.pub_brake.publish(brake_msg)
        # publish brake_msg

        brake_bool_msg.data=False
        self.pub_brake_bool.publish(brake_bool_msg)
```

==========

2. Wall Follower

Yotube Link: https://youtu.be/vpMMhKbd5ac

1. Concept

1. PID Control

- PID Control은 시스템의 특정 변수를 원하는 값 주변으로 유지하는 제어 방법이다.
- 본 프로젝트에서 제어하는 변수[u(t)]는 Steering Angle이며 e(t)는 t에서 벽과 Vehicle 사이의 거리와 원하는 거리의 차이이다.
- Trial and Error를 통해 적절한 P, I, D Gain을 구한다. 본 프로젝트에서는 I Control은 제외했다.

$$u(t) = K_p\, e(t) + K_i \int_0^t e(t')\, dt' + K_d \frac{d}{dt}(e(t))$$

<FIG.5 PID Control Equation>

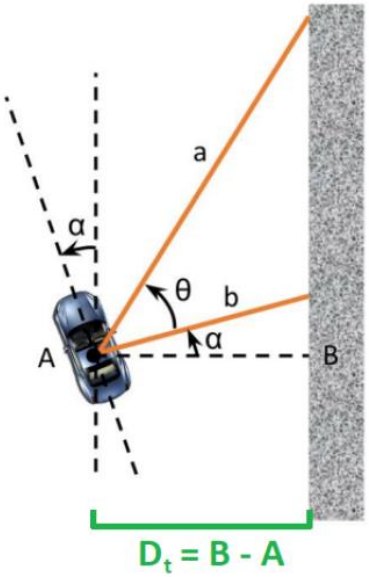
2. Wall Follower

- Scan Topic의 LaserScan 정보를 통해 t에서 벽과 Vehicle의 거리를 구한다.
- 오른쪽 벽으로 분사하는 임의의 Laser Point 두 개를 이용한다. 두 개의 Laser point 중 더 먼 것의 range를 a, 가까운 것을 b, 사이각을 theta로 설정한다.
- a, b, theta를 통해 Vehicle의 x축에서의 벽과의 거리를 구한다.

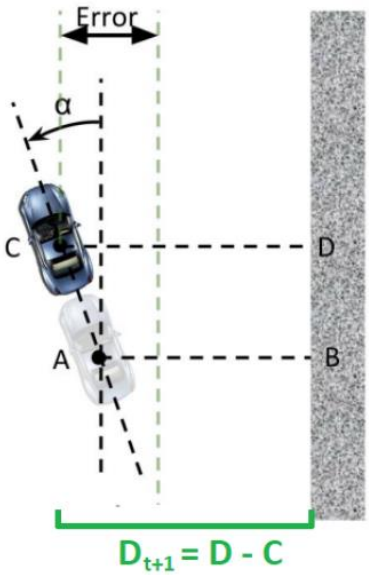
$$\alpha = \tan^{-1}\frac{a\sin\theta - b}{a\sin\theta}\ ,\ D_t = b\cos\alpha$$

- Error에서 t에서 Vehicle과 벽과의 거기를 사용하면 반응 속도가 느리므로 t+1에서의 Vehicle과 벽과의 거리를 이용한다.

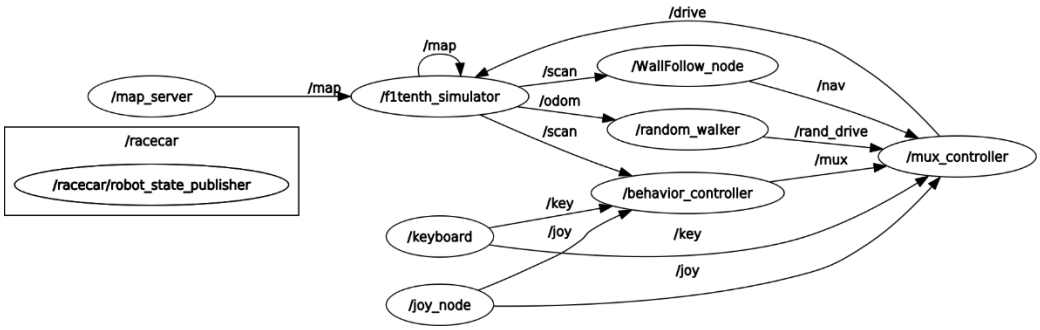
$$D_{t+1} = D_t + L\sin\alpha$$



<FIG.6 Distance and orientation of vehicle relative to wall>



<FIG.7 Future Distance>



<FIG.8 Wall Follower rqt_graph>

2. Algorithm

1. def __init__(self)

- Node를 실행할 때 Subscribe할 토픽의 이름, Type, Callback 함수를 설정하고 Publish할 토픽의 이름, Type, queue_size를 설정한다.
- 해당 노드에서는 f1tenth_simulator에서 scan 토픽을 Subscribe하여 Vehicle 과 벽 사이의 거리를 계산한다.
- 또한, PID Control을 통한 Steering Angle을 nav 토픽으로 Publish한다.

==========

```
def __init__(self):

        lidarscan_topic = '/scan'
        drive_topic = '/nav'
```

```
   # node subscribes to the scan topic which is of type sensor_msgs/LaserScan.
        self.lidar_sub = rospy.Subscriber(lidarscan_topic, LaserScan,
self.lidar_callback)
        # node is publishing to the /brake topic using the message type
AckermannDrivestamped
        self.drive_pub = rospy.Publisher(drive_topic, AckermannDriveStamped,
queue_size=10)
```

==========

2. def lidar_callback(self, data)

- followRight 함수를 통해 scan 토픽의 LaserScan 데이터를 이용하여 error를 구한다.
- Pid_control 함수를 통해 error와 Vehicle의 속도를 이용하여 Steering Angle 값을 설정한다.

==========

```
def lidar_callback(self, data):

        laser_data = data

        error = self.followRight(laser_data, DESIRED_DISTANCE_LEFT)

        self.pid_control(error, VELOCITY)
        # set the steering angle for the VESC of vehicle using PID Control
```

==========

3. def followRight(self, data, RightDist)

- getRange 함수를 이용하여 Vehicle과 오른쪽 벽 임의의 Laser Point 2개 사이의 거리를 구한다. 두 개의 Laser Point 사이의 각도는 THETA이다.
- 두 개의 거리와 THETA를 이용하여 α를 구하고 현재와 미래에서의 Vehicle의 x축에서의 벽과의 거리를 구한다.
- 사전에 THETA는 pi/10으로 설정했다.

==========

```
def followRight(self, data, RightDist):

        a, b = self.getRange(data, THETA)
 # get the ditstance between vehicle and two random points of right wall

        alpha = atan((a*cos(THETA)-b) / a*sin(THETA))
        # get the angle between two distance vector

        D_cur = b*cos(alpha)
        # current distance to wall from ego vehicle
        D_fut = D_cur + CAR_LENGTH*sin(alpha)
        # future distance to wall from ego vehicle

        caculated_error = D_fut - RightDist
        # set error used in PID control
        return caculated_error
```

==========

4. def getRange(self, data, angle)

- Vehicle에서 오른쪽 벽에 두개의 Laser Point를 분사하고 거리 값을 얻는다. Laser Pointdml 각도 차이는 angle이다. Vehicle의 x축에 가까운 거리를 a, 먼 거리를 b라 한다.
- 본 프로젝트에서는 a에 해당하는 Laser Point는 angle_min으로부터 $269 \times \frac{270}{1080}$ 도 떨어진 각도이고 b에 해당하는 Laser Point는 $269 \times \frac{270}{1080}$ +angle이다.
- 두 개의 Laser Point에 해당하는 거리 a, b를 Return 한다.

==========

```
def getRange(self, data, angle):                    # get the ditstance between vehicle and two random points of right wall

        b = data.ranges[269]

        i = int(angle / (2*pi/1080))

        a = data.ranges[269+i]

        return a,b
```

==========

5. def pid_control(self, error, velocity)

- P Control에 입력되는 error는 미래의 Vehicle의 오른쪽 벽 사이 거리와 원하는 Vehicle과 벽 사이의 거리이다.
- D control에 입력되는 error는 현재 error와 미래의 error의 차이이다.
- PID control의 출력은 Steering Angle이다.
- AckermannDriveStamped 형식의 메시지를 생성한 후 Steering angle 변수에 PID Control 출력을 입력한 뒤 drive_topic Topic으로 Publish한다.

==========

```
def pid_control(self, error, velocity):
        global integral
        global prev_error
        global kp
        global ki
        global kd
        angle = 0.0

        angle = kp * error + kd * (error - prev_error)

        drive_msg = AckermannDriveStamped()
        # message which type is AckermannDriveStamped
        drive_msg.header.frame_id = "laser"
        # update parameter frame_id in drive_msg as "laser"
```

```
        drive_msg.drive.steering_angle = -angle
        # update parameter steering_angle in drive_msg as -angle
        drive_msg.drive.speed = velocity
        # update parameter speed in drive_msg as velocity

        self.drive_pub.publish(drive_msg)
        # publish drive_msg

        print(error)
        prev_error = error                          # set previous error
================================================================
```