

LiDAR 실습 과제

Department of Electrical Engineering, Incheon National University
Hwasu Lee

2024.07.02~04

1. LiDAR 센서 기반 정지 알고리즘 구현

1. LiDAR 센서 기반 정지 알고리즘 구현

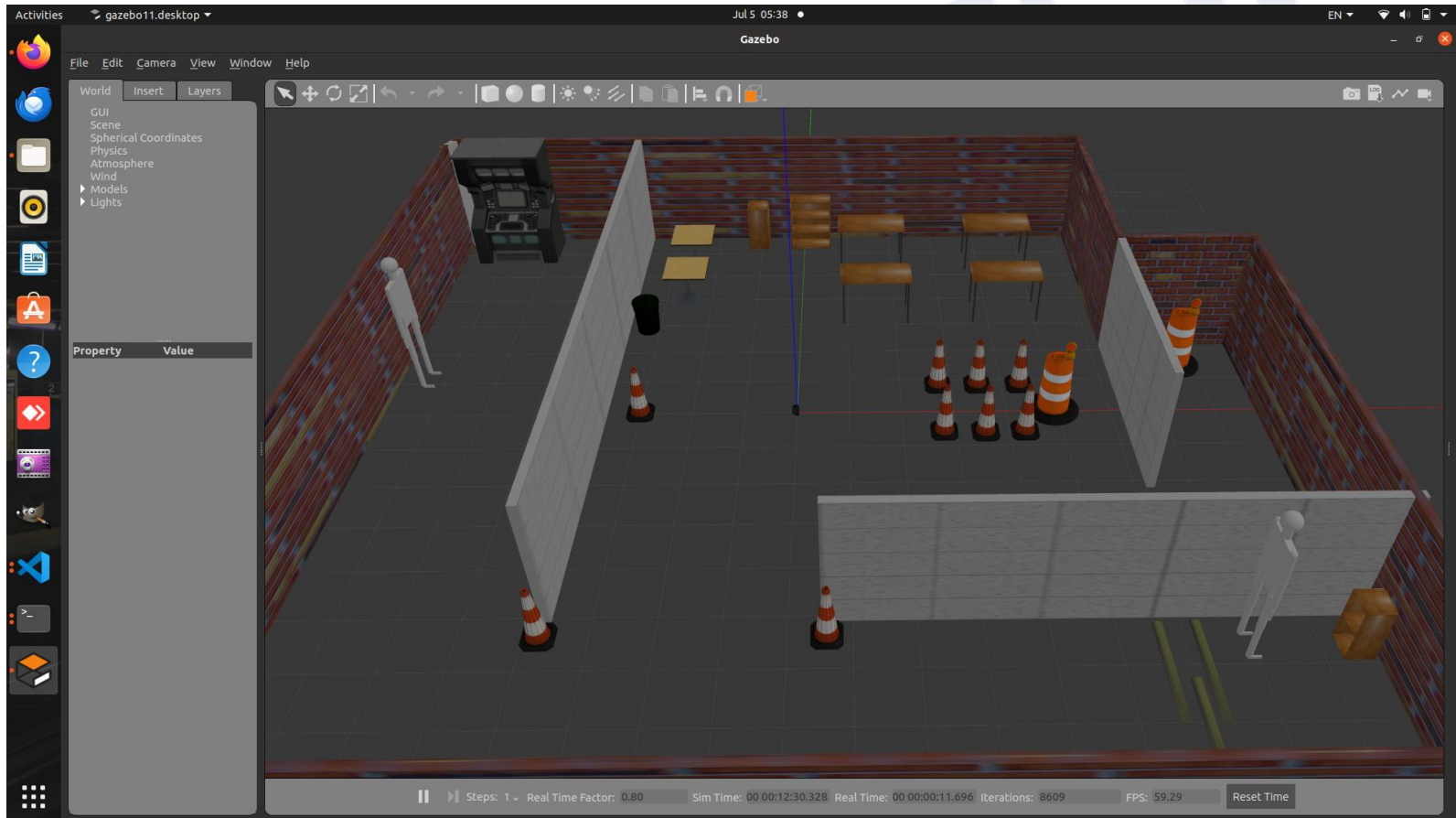
- `$ roscd turtlebot_stop_cmd/scripts`
- `$ gedit stop2.py` (코드 작성 → 코드 작성 가이드 뒷장에 설명)
- `$ sudo chmod +x stop2.py` (실행 권한 부여)
- `$ roscd turtlebot3_gazebo/worlds`
- `$ gedit custom_world.world` (첨부된 world 파일 복사 및 붙여넣기 후 저장)
- `$ roscd turtlebot3_gazebo/launch`
- `$ cp turtlebot3_empty_world.launch turtlebot3_custom_world.launch`
- `$ gedit turtlebot3_custom_world.launch` (코드 수정 → 코드 수정 가이드 뒷장에 설명)

→ 각각의 자세한 설명은 뒷장에

1. LiDAR 센서 기반 정지 알고리즘 구현

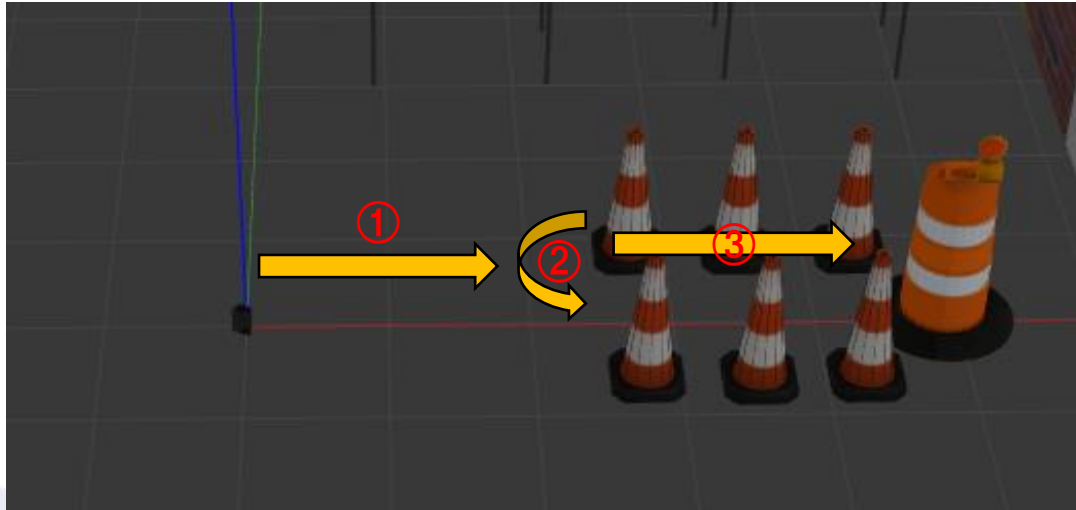
- stop2.py 코드 작성 가이드

실습 : 어제 수업했던 LiDAR 센서 기반 정지 알고리즘을 바탕으로, 유사 정지(주차) 알고리즘 구현



1. LiDAR 센서 기반 정지 알고리즘 구현

- stop2.py 코드 작성 가이드



- \$ roscd turtlebot_stop_cmd/scripts
- \$ gedit stop2.py (코드 작성 → 코드 정답은 일요일 자정에 공유 예정)
 - 전진 과정에서 전방 LiDAR 데이터가 0.5m 이하가 되면, 정지 후 180도 제자리 회전 수행
 - 제자리 회전이 끝나면, 뒤로 주행을 시작하여 가장 큰 라바콘과의 거리가 0.2m 이하가 되면
정지하는 코드 구현
- \$ sudo chmod +x stop2.py (실행 권한 부여)

1. LiDAR 센서 기반 정지 알고리즘 구현

- custom_world.world 작성 가이드

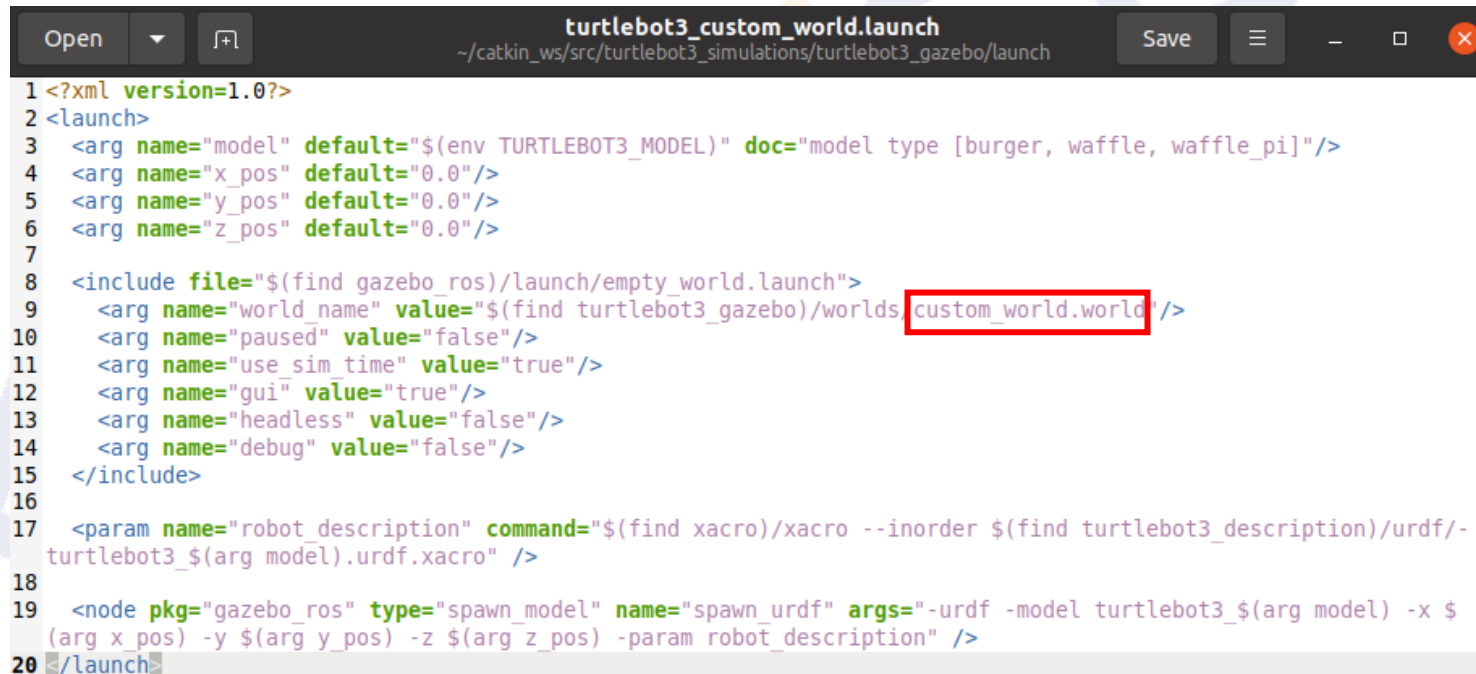
- 1) 첨부한 “custom_world.world” 파일 내용을 복사
- 2) `$ roscd turtlebot3_gazebo/worlds` (이동)
- 3) `$ gedit custom_world.world` (복사한 world 파일 내용을 붙여넣기 후 저장)
- 4) `$ roscd turtlebot3_gazebo/launch` (이동)
- 5) `$ cp turtlebot3_empty_world.launch turtlebot3_custom_world.launch` (복사)
- 6) `$ gedit turtlebot3_custom_world.launch` (파일 열기)

→ `turtlebot3_custom_world.launch` 파일 수정 방법은 뒷장에 계속...

1. LiDAR 센서 기반 정지 알고리즘 구현

- custom_world.world 작성 가이드

1) \$ gedit turtlebot3_custom_world.launch (파일 열기)



```
1 <?xml version=1.0?>
2 <launch>
3   <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
4   <arg name="x_pos" default="0.0"/>
5   <arg name="y_pos" default="0.0"/>
6   <arg name="z_pos" default="0.0"/>
7
8   <include file="$(find gazebo_ros)/launch/empty_world.launch">
9     <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/custom_world.world" />
10    <arg name="paused" value="false"/>
11    <arg name="use_sim_time" value="true"/>
12    <arg name="gui" value="true"/>
13    <arg name="headless" value="false"/>
14    <arg name="debug" value="false"/>
15  </include>
16
17  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/-
  turtlebot3_${arg model}.urdf.xacro" />
18
19  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_${arg model} -x $
  (arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
20 </launch>
```

위 이미지의 빨간색 박스와 같이 world 파일 수정 후 저장 !!

(empty_world.world → custom_world.world)

1. LiDAR 센서 기반 정지 알고리즘 구현

→ 만약 새로운 world 파일 수정 및 LiDAR 센서 기반 정지 알고리즘 구현이 완료되면,

- `$ cd ~/catkin_ws && catkin_make`
(catkin workspace build)
- `$ source devel/setup.bash && source /opt/ros/noetic/setup.bash`
(environment setup)
- `$ (Terminal 1에서 실행) export TURTLEBOT3_MODEL=burger`
(define model)
`roslaunch turtlebot3_gazebo turtlebot3_custom_world.launch`
(execute turtlebot3 simulation environment)
- `$ (Terminal 2에서 실행) rosrun turtlebot_stop_cmd stop2.py`
(execute turtlebot3 driving)

1. LiDAR 센서 기반 정지 알고리즘 구현

→ 참고 자료

Measurement Performance Specifications

Items	Specifications
Distance Range	120 ~ 3,500mm
Distance Accuracy (120mm ~ 499mm)	±15mm
Distance Accuracy(500mm ~ 3,500mm)	±5.0%
Distance Precision(120mm ~ 499mm)	±10mm
Distance Precision(500mm ~ 3,500mm)	±3.5%
Scan Rate	300±10 rpm
Angular Range	360°
Angular Resolution	1°

- https://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html → 속도 관련 메시지 정보

Turtlebot3 LiDAR range & resolution

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

→ ROS 1 Noetic 환경에서 LiDAR 기반 정지 알고리즘 구현에 필요한 Python 및 ROS 라이브러리 설치

```
1  #!/usr/bin/env python
2
3  import rospy
4  from sensor_msgs.msg import LaserScan
5  from geometry_msgs.msg import Twist
6  from nav_msgs.msg import Odometry
7  from tf.transformations import euler_from_quaternion
8  import math
```

- `import rospy` : ROS 환경에서 python 기반 노드 작성에 필수적인 라이브러리 import
- `from sensor_msgs.msg import LaserScan` : LiDAR scan 데이터를 Subscribe 받기 위해 import
- `from geometry_msgs.msg import Twist` : Turtlebot의 속도 명령을 Publish 해주기 위해 import
- `from nav_msgs.msg import Odometry` : Turtlebot의 Odometry 정보를 바탕으로 정확한 회전을 위해 import
- `from tf.transformations import euler_from_quaternion` : Turtlebot의 Odometry 정보로부터 도출된 orientation(x,y,z,w) 값을 yaw 단위로 변환하기 위해 import

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```

10 class StopAndParkingNode:
11     def __init__(self):
12         # 현재 작성중인 노드의 이름 정의
13         rospy.init_node('stop_and_parking_node', anonymous=True)
14
15         # Publisher(속도)와 Subscriber(LiDAR 데이터, Odometry 데이터) 정의
16         self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
17         self.scan_sub = rospy.Subscriber('/scan', LaserScan, self.scan_callback)
18         self.odom_sub = rospy.Subscriber('/odom', Odometry, self.odom_callback)
19
20         self.stop_distance = 2.0          # 정지해야 할 거리 (전방 2.0m)
21         self.stop_distance_2 = 0.5        # 정지해야 할 거리 (후방 0.5m)
22         self.linear_speed = 0.2           # 직진 속도 (m/s)
23         self.angular_speed = 0.2          # 회전 속도 (rad/s)
24         self.turn_flag = False             # 전진 후, 회전을 시작하기 위한 boolean
25         self.rear_flag = False             # 회전 후, 후진을 시작하기 위한 boolean
26         self.current_angle = 0.0           # 현재 로봇의 각도
27         self.target_angle = None           # 로봇이 회전해야 하는 목표 각도
  
```

- StopAndParkingNode라는 이름의 Class를 생성 후, Publisher와 Subscriber 정의
- 또한 생성한 Class의 내부에서 사용할 여러 변수들을 정의

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```
30 """
31 | 로봇의 LiDAR 데이터를 지속적으로 불러올 수 있도록 하는 callback 함수
32 | 해당 callback 함수를 통해 전,후방 LiDAR 데이터를 지속적으로 불러올 수 있음
33 """
34 def scan_callback(self, msg):
35
36     # LiDAR 데이터의 중앙 인덱스 계산
37     center_index = len(msg.ranges) // 2
38
39     # 전방 5개의 LiDAR 데이터 추출
40     front_distances = msg.ranges[:5] + msg.ranges[-5:]
41
42     # 후방 5개의 LiDAR 데이터 추출
43     rear_distances = msg.ranges[center_index - 5 : center_index + 6]
44
45     # 해당 범위 내 가장 가까운 거리 확인
46     closest_front_distance = min(front_distances)
47     closest_rear_distance = min(rear_distances)
48
```

- LiDAR의 Scan data를 Subscribe 받을 때마다, 지속적으로 실행될 scan_callback 함수 구성
- 해당 함수에서는 scan data를 기반으로 전/후방 데이터를 추출하여 전진/회전/후진을 수행할 수 있도록 설계 필요

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```

49 # 만약 회전 flag와 후진 flag가 모두 False인 경우, 전진 또는 정지하도록 코드 구성
50 if (self.turn_flag == False) and (self.rear_flag == False):
51
52     if closest_front_distance < self.stop_distance:
53         # 거리가 설정값보다 짧으면 정지 명령을 보냄
54         rospy.loginfo("Obstacle detected in front. Stopping the robot.")
55         self.stop_robot()
56
57         # 로봇을 회전할 수 있도록 하는 boolean을 True로 변경
58         self.turn_flag = True
59         rospy.sleep(0.5)
60
61     else:
62         # 거리가 설정값보다 길면 직진 명령을 보냄
63         rospy.loginfo("Closest distance in front 10 degrees: {:.2f} m".format(closest_front_distance))
64         self.move_forward()
  
```

- 먼저 회전이나 후진을 수행하지 않고, 일정 거리 이내로 전진할 수 있도록 조건문 설정
- 해당 조건문에서는 전방 LiDAR 데이터를 활용하여 2.0m 이내로 주행 후, 정지할 수 있도록 설계 필요

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```

66     # 만약 회전 flag가 True인 경우, 제자리 180도 회전을 할 수 있도록 코드 구성
67     elif (self.turn_flag == True) and (self.rear_flag == False):
68         self.target_angle = 3.14 # 로봇이 회전해야 하는 목표 각도 설정 (단위: radian)
69         # 정지 후, 180도 회전 명령을 보냄
70         rospy.loginfo("Robot is rotating...")
71
72         while not rospy.is_shutdown():
73             angle_diff = self.target_angle - self.current_angle
74
75             # 회전 오차를 고려하여 로봇이 180도 회전한 경우, 회전을 정지 !
76             if abs(angle_diff) < 0.02: # 오차 허용 범위 내 도달 시 정지 (단위: radian)
77                 # 로봇이 180도 회전을 마치면 잠시 정지시킨 뒤, 후진하기 위한 boolean 값을 True로 변경
78                 self.rear_flag = True
79                 self.stop_robot()
80                 break
81
82             self.turn_robot()
83             rospy.sleep(0.1)

```

- 다음 조건문으로 Turtlebot3 모델이 2.0m 이내로 주행한 뒤 정지하였을 때, 해당 위치에서 제자리 180도 회전을 수행할 수 있도록 코드 설계 필요
- 이 때, Turtlebot3 모델의 Odometry(주행기록계) 정보를 활용하여, 현재 Turtlebot3 모델의 각도인 `self.current_angle` 값이 `self.target_angle = 3.14(180도)` 값과 일정 오차 범위를 포함하여 같아질 때 까지 제자리 회전 수행할 수 있도록 코드 설계

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```
85     # 만약 로봇이 180도 제자리 회전을 마쳐 후진 flag가 True가 된 경우, 후진할 수 있도록 코드 구성
86     elif (self.rear_flag):
87         # 180도 회전을 마친 후, 로봇이 후진하도록 속도 명령을 보냄
88         if closest_rear_distance > self.stop_distance_2:
89             rospy.loginfo("Closest distance in rear 10 degrees: {:.2f} m".format(closest_rear_distance))
90             self.move_back()
91         else:
92             self.stop_robot()
93
94     # Error 방지
95     else:
96         rospy.loginfo("Code Error")
```

- 마지막 조건문으로 Turtlebot3 모델이 180도 제자리 회전을 수행한 경우, 후방 LiDAR 데이터를 통해 0.5m 이내로 도달할 때 까지 후진 주행을 수행할 수 있도록 코드 설계 필요
- 만약, Turtlebot3 모델이 0.5m 이내로 후진 주행 완료 시, 정지할 수 있도록 코드 설계 필요

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```

98  """
99  로봇의 Odometry 데이터를 지속적으로 불러올 수 있도록 하는 callback 함수
100  해당 callback 함수를 통해 로봇의 주행기록계(위치 및 방향) 정보를 지속적으로 불러올 수 있으며,
101  이를 통해 로봇의 현재 각도 정보를 알 수 있음
102  """
103  # 로봇의 주행기록계(위치 및 방향)를 통해 로봇의 현재 각도 정보를 받아오기
104  def odom_callback(self, msg):
105      orientation_q = msg.pose.pose.orientation
106      orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w]
107      _, _, yaw = euler_from_quaternion(orientation_list)
108      self.current_angle = yaw
109      #print(math.degrees(self.current_angle)) # 로봇의 각도를 radian에서 degree로 변경 후, 터미널에 출력 (단위: degree)
  
```

- 해당 메서드는 Turtlebot3의 Odometry 정보를 Subscribe 받을 때마다, 수행될 수 있도록 하는 call_back 함수
- 해당 메서드에서는 Turtlebot3의 주행 기록계 (위치 및 방향) 정보를 통해, 현재 Turtlebot3 모델의 yaw 값이 얼마인지 측정하는 과정을 수행

2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```

112     """
113     로봇의 실제 전진/정지/회전/후진 명령을 생성하도록 하는 메서드 정의
114     """
115     # 로봇이 정지하도록 속도 명령 생성
116     def stop_robot(self):
117         # 로봇을 정지시키는 메시지 생성
118         stop_msg = Twist()
119         stop_msg.linear.x = 0.0
120         stop_msg.angular.z = 0.0
121         self.cmd_vel_pub.publish(stop_msg)
122
123     # 로봇이 전진하도록 속도 명령 생성
124     def move_forward(self):
125         # 로봇을 직진시키는 메시지 생성
126         move_msg = Twist()
127         move_msg.linear.x = self.linear_speed
128         move_msg.angular.z = 0.0
129         self.cmd_vel_pub.publish(move_msg)
130
131     # 로봇이 회전하도록 속도 명령 생성
132     def turn_robot(self):
133         move_msg = Twist()
134         move_msg.linear.x = 0.0
135         move_msg.angular.z = self.angular_speed
136         self.cmd_vel_pub.publish(move_msg)
137
138     # 로봇이 후진하도록 속도 명령 생성
139     def move_back(self):
140         move_msg = Twist()
141         move_msg.linear.x = -self.linear_speed
142         move_msg.angular.z = 0.0
143         self.cmd_vel_pub.publish(move_msg)
  
```

- 해당 메서드들은 Turtlebot3 모델의 정지, 주행, 회전, 후진 등의 제어 명령을 publish 하기 위해 정의한 메서드
- 초기 import한 `geometry_msgs.msg`의 `Twist`라는 메시지 타입을 기반으로 각 메서드에 맞는 적절한 제어 입력을 publish

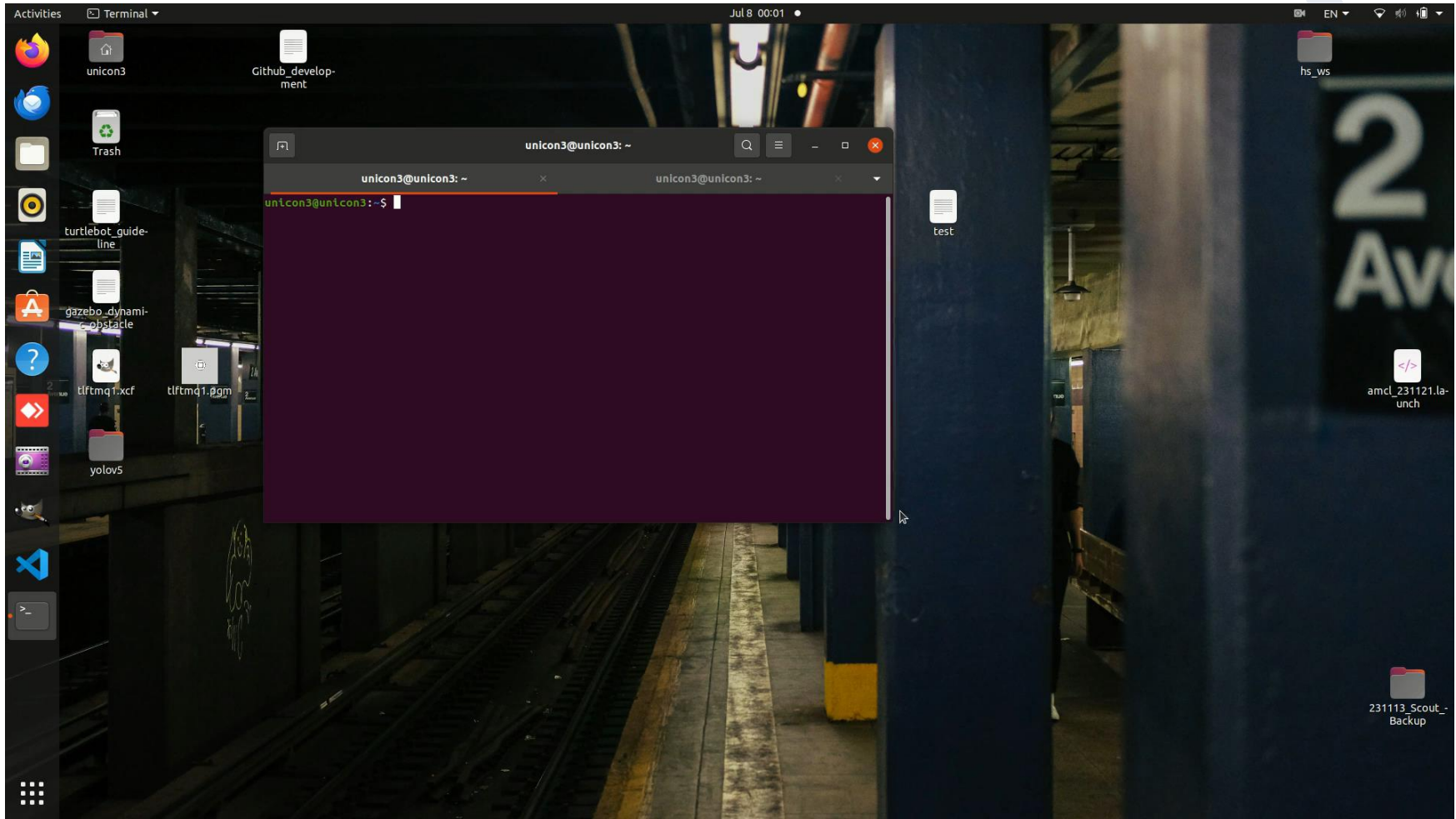
2. LiDAR 센서 기반 정지 알고리즘 코드 리뷰

```
145     # 코드를 지속적으로 반복
146     def run(self):
147         rospy.spin()
148
149     if __name__ == '__main__':
150         node = StopAndParkingNode()
151         # 데이터를 안전하게 받아올 수 있게 잠시 코드를 정지
152         rospy.sleep(1)
153         node.run()
```

- run 메서드 내부의 rospy.spin()은 해당 노드가 종료되기 전까지 지속적으로 노드를 반복 실행될 수 있도록 하는 rospy의 내장 함수
- spin() 함수는 'ctrl+c' 또는 'rospy.signal_shutdown()' 등의 종료 신호가 들어오는 경우 반복 루프를 종료하고 노드가 정상적으로 종료될 수 있도록 하며, 이 외의 경우는 설계한 노드가 지속적으로 반복할 수 있도록 함
- 이후, 설계한 class를 선언하고 반복 실행하기 위한 run 메소드를 실행함으로써, 해당 노드에서 설계한 call_back 함수들을 호출하여 노드가 적절하게 동작할 수 있도록 함

3. LiDAR 센서 기반 정지 알고리즘 구현 영상

3. LiDAR 센서 기반 정지 알고리즘 구현 영상



QnA

E-mail : leehwasu9696@inu.ac.kr

Mobile : 010-8864-5585



청주대학교
미래형자동차 인력양성사업단

자율주행 분야

단기집중 교육과정



6월 11일(화)
오후 4시~8시

리눅스 설치,
ROS 설치 및 ROS 개발환경 구축

융합관
410호

*개인 노트북 필수 지참(대여불가)

온라인
사전교육

Python 교육(ROS 기반)



7월 2일~
5일(화~금)
오후 1시~6시

라이다 센서 교육

새천년종합정보관
107A호

경진대회 참여학생

7월 8일~
10일(월~수)
오후 1시~6시

ROS 활용 교육(Python 기반)

융합관
410호

*개인 노트북 필수 지참(대여불가)

7월 15일~
17일(월~수)
오후 1시~6시

Python OpenCV 설치 및 응용

새천년종합정보관
409호

*개인 노트북 필수 지참(대여불가)



Thank you.

