**스마트모빌리티설계**

**[과제4] LiDAR - Motor구동 과제**

1. **코드**

#!/usr/bin/env python

import rospy, time, random

from sensor\_msgs.msg import LaserScan

from xycar\_msgs.msg import xycar\_motor

# CONSTANT

SCAN\_ANGLE\_RANGE = 23 # 23degree

DISTANCE\_THRESHOLD = 0.65 # 0.6m

SPEED = 3 # 0~5

ANGLE = 35 # 0~50(absolute value)

class Car():

def \_\_init\_\_(self):

rospy.init\_node('lidar\_driver')

rospy.Subscriber('/scan', LaserScan, self.callback, queue\_size=1)

self.pub = rospy.Publisher('xycar\_motor', xycar\_motor, queue\_size=1)

self.motor\_msg = xycar\_motor()

self.lidar\_points = None

def callback(self, data):

self.lidar\_points = data.ranges

def drive(self, speed, angle):

self.motor\_msg.speed = speed

self.motor\_msg.angle = angle

self.pub.publish(self.motor\_msg)

def obstacle\_detected(self):

count = 0

for degree in range(0, 2 \* SCAN\_ANGLE\_RANGE):

if 0.01 < self.lidar\_points[degree] <= DISTANCE\_THRESHOLD:

count += 1

if 0.01 < self.lidar\_points[719 - degree] <= DISTANCE\_THRESHOLD:

count += 1

return count > 5

def avoid\_obstacle(self):

direction = random.choice(["left", "right"])

angle = ANGLE if direction == "right" else -ANGLE

rospy.loginfo("[Avoiding] direction: " + direction)

# 1. L or R

for i in range(15):

self.drive(SPEED, angle)

time.sleep(0.1)

# 2. straight

for i in range(10):

self.drive(SPEED, 0)

time.sleep(0.1)

# 3. R or L

for i in range(30):

self.drive(SPEED, -angle)

time.sleep(0.1)

# 4. straight

for i in range(10):

self.drive(SPEED, 0)

time.sleep(0.1)

# 5. L or R

for i in range(15):

self.drive(SPEED, angle)

time.sleep(0.1)

rospy.loginfo("Avoid complete")

def main():

car = Car()

while car.lidar\_points is None:

continue

cleared = None

while not rospy.is\_shutdown():

if car.obstacle\_detected():

rospy.loginfo("Obstacle detected! Stop!")

car.drive(0, 0)

start\_time = time.time()

cleared = False

while time.time() - start\_time < 5:

if not car.obstacle\_detected():

cleared = True

rospy.loginfo("Obstacle eliminated! Go straight!")

car.drive(SPEED, 0)

break

if not cleared:

rospy.loginfo("5sec over. Avoid")

car.avoid\_obstacle()

else:

rospy.loginfo("No obstacles! Go straight!")

car.drive(SPEED, 0)

if \_\_name\_\_ == "\_\_main\_\_":

try:

main()

except rospy.ROSInterruptException:

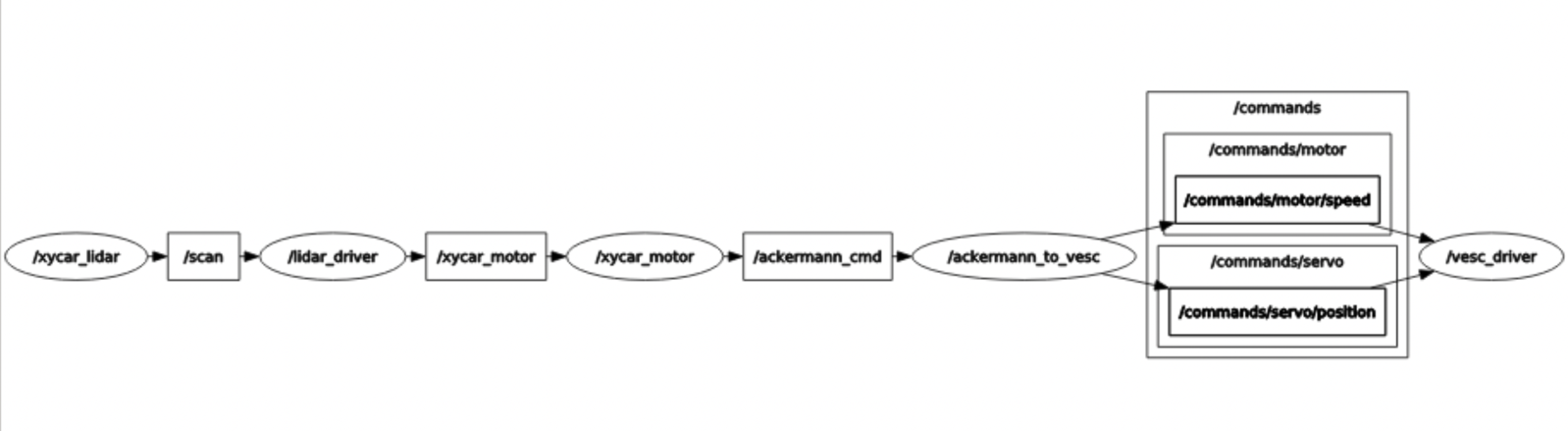
pass

1. **코드 설명**
   1. **CONSTANT**
      1. **SCAN\_ANGLE\_RANGE**
         1. LiDAR로부터 얻은 거리값 중 사용할 범위의 각도(degree)를 나타내는 상수입니다. 정면을 기준으로 좌우 각각 SCAN\_ANGLE\_RANGE/2 deg씩 사용합니다.
      2. **DISTANCE\_THRESHOLD**
         1. 장애물을 탐지할 거리의 임계값을 저장하는 상수입니다. 이 거리보다 가까운 거리에서 장애물이 탐지된다면 장애물이 존재한다고 판단합니다.
      3. **SPEED**
         1. 모터를 구동할 속도를 저장하는 상수입니다. 0~5의 범위를 가질 수 있습니다.
      4. **ANGLE**
         1. 앞바퀴 조향 각을 저장하는 상수입니다. -50~50의 절대값으로, 0~50의 범위를 가질 수 있습니다.
   2. **Car class 구조**
      1. **\_\_init\_\_() 함수**
         1. Car 객체 생성 시 초기화를 하는 함수입니다. lidar\_driver node를 초기화하고, LiDAR data를 얻기 위해 /scan topic을 subscribe하며, /xycar\_motor topic으로 모터 구동을 정보를 보내기 위한 publisher를 등록합니다. 또한, 모터 제어 정보를 전송하기 위한 msg와 LiDAR 정보를 저장할 변수를 초기화합니다.
      2. **callback() 함수**
         1. LiDAR 정보가 들어올 때마다 실행되는 함수이며, 들어온 정보를 self.lidar\_points에 저장합니다.
      3. **drive() 함수**
         1. 인자로 전달 받은 속도와 조향 각으로 자동차를 제어하는 함수입니다. 전달 받은 속도와 조향 각을 self.motor\_msg에 담아 xycar\_motor topic로 publish합니다.
      4. **obstacle\_detected() 함수**
         1. 장애물이 탐지되었는지 판단하는 함수이며, LiDAR 데이터의 유효하게 사용하는 범위 내에 DISTANCE\_THRESHOLD보다 가까운 거리에 장애물이 존재하는지 판단합니다.
      5. **avoid\_obstacle() 함수**
         1. 장애물을 회피하고 원래의 진행 경로로 복귀하도록 모터를 제어하는 함수입니다. 좌우 중 random하게 선택한 방향으로 회피합니다. 회피한 왼쪽으로 움직인 양과 오른쪽으로 움직인 양을 동일하게 하여 원래 경로로 복귀하도록 구현하였습니다.
   3. **main() 함수**
      1. Car 객체를 생성하고, 전진 주행을 하다가 장애물이 탐지된 경우(car.obstacle\_detected()) 정지한 상태로 기다리면서 장애물이 제거되었는지 확인합니다. 5초 안에 장애물이 제거된 경우 전진 주행을 하며, 5초동안 장애물이 제거되지 않은 경우 장애물을 회피하여 주행하는 함수(car.avoid\_obstacle())를 호출합니다.
2. **결과 분석**

위 코드를 실행한 결과, 자동차가 진행 중 설정한 거리(0.65m) 내에 전방에 장애물이 발견되면 정지한 상태로 기다리면서 5초가 되기 전에 장애물이 제거되면 다시 전진하고 장애물이 제거되지 않는 경우 회피 주행을 성공적으로 하는 모습을 확인할 수 있었습니다. 또한 장애물을 회피할 뿐 아니라, 원래 진행 경로로 성공적으로 복귀하는 것을 확인할 수 있었습니다.

자동차가 장애물을 회피하면서 장애물과 충돌하지 않도록 장애물 탐지 거리(DISTANCE\_THRESHOLD)를 0.65m로 실험적으로 결정하였습니다.

rqt\_graph를 통해서도 각 node가 올바르게 통신을 하고 있는 것을 확인할 수 있습니다. /xycar\_lidar node가 publish한 거리 데이터를 /lidar\_driver node가 subscribe하고, /lidar\_driver node가 publish한 모터 제어 정보를 /xycar\_motor node가 subscribe하여 목표대로 잘 통신하고 있음을 확인하였습니다.

****

1. **고찰**
   1. 회피 전 IMU의 yaw 값 저장한 후 원래 경로로 복귀할 때 활용하면 오차를 줄이고 더 정확하게 구현할 수 있을 것입니다.
   2. 회피 방향을 좌우 중 random하게 결정하였지만, 회피 방향을 LiDAR 스캔 범위 내 가장 멀리 떨어진 방향으로 결정한다면 더 효율적으로 회피할 수 있을 것입니다.
   3. 회피하는 도중에는 장애물을 탐지하지 않도록 구현하였지만, 회피 중에도 장애물을 탐지한다면 더 안전한 주행이 가능할 것입니다.
   4. 주행에 있어서는 속도와 조향 각을 고정하였지만, 상황에 따라 동적으로 조절한다면 더 부드러운 주행이 가능할 것입니다.