LIMO 라이다 주행

목 차

- 1. Lidar 센서를 통한 장애물 인식
- 2. Lidar 센서를 통한 터널주행



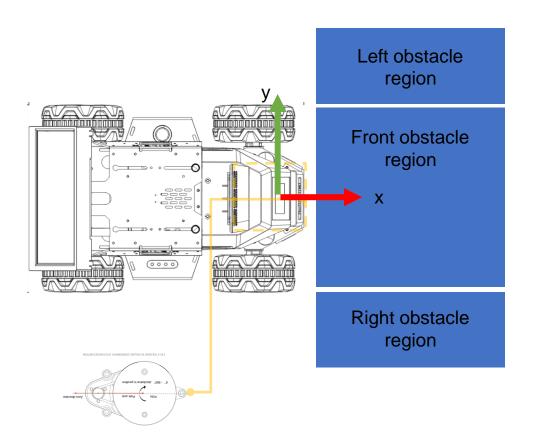
01 1. Lidar 센서를 통한 장애물 인식

- ydlidar 준비
 - -. ydlidar_ros_driver 실행
 - \$ roslaunch ydlidar_ros_driver X2.launch
 - -. X2.launch 파일 확인
 - : 라이다 좌표 reversion 'true'
 - : 라이다 측정 범위 -100~100

```
<node name="ydlidar_lidar_publisher" pkg="ydlidar_ros_driver" type="ydlidar</pre>
respawn="false" >
   <!-- string property -->
                                 type="string" value="/dev/ydlidar"/>
   <param name="port"</pre>
                                 type="string" value="laser frame"/>
   <param name="frame id"</pre>
   <param name="ignore array"</pre>
                                     type="string" value=""/>
   <!-- int property -->
   <param name="baudrate"</pre>
                                     type="int" value="115200"/>
   <!-- 0:TYPE TOF, 1:TYPE TRIANGLE, 2:TYPE TOF NET -->
   <param name="lidar_type"</pre>
                                     type="int" value="1"/>
   <!-- 0:YDLIDAR TYPE SERIAL, 1:YDLIDAR TYPE TCP -->
   <param name="device_type"</pre>
                                        type="int" value="0"/>
   <param name="sample rate"</pre>
                                        type="int" value="3"/>
   <param name="abnormal_check_count"</pre>
                                                  type="int" value="4"/>
   <!-- bool property -->
   <param name="resolution_fixed"</pre>
                                        type="bool" value="true"/>
   <param name="auto reconnect"</pre>
                                     type="bool" value="true"/>
   <param name="reversion"</pre>
   <param name="inverted"</pre>
                                               value="true"/>
                                       type="bool" value="true"/>
   <param name="isSingleChannel"</pre>
   <param name="intensity"</pre>
                                 type="bool"
                                               value="false"/>
   <param name="support motor dtr"</pre>
                                         type="bool" value="true"/>
   <param name="invalid_range_is_inf"</pre>
                                            type="bool" value="false"/>
   <param name="point cloud preservative"</pre>
                                                type="bool" value="false"/>
   <!-- float property -->
   <param name="angle_min"</pre>
                                 type="double" value="-90" />
                                 type="double" value="90" />
   <param name="angle max"</pre>
   <param name="range min"</pre>
                                 type="double" value="0.1" />
   <param name="range_max"</pre>
                                 type="double" value="12.0" />
   <!-- frequency is invalid, External PWM control speed -->
                                type="double" value="10.0"/>
```

01 1. Lidar 센서를 통한 장애물 인식

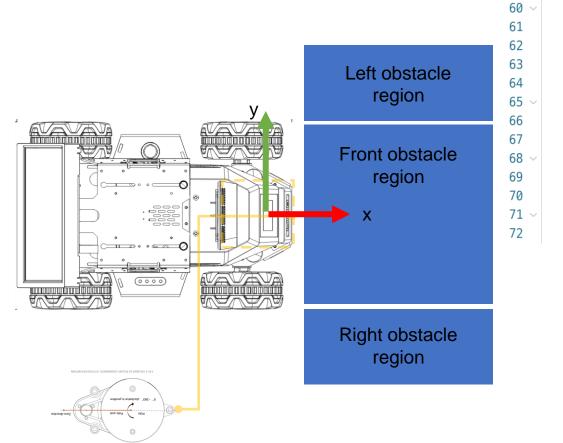
- 라이다를 통한 장애물 인식 실습
 - -. 장애물 인식(판단) 알고리즘



```
def lidar CB(self, msg):
   if msg != -1: # 유효한 메시지가 들어왔을 경우:
       self.laser msg = msg # LaserScan 메시지를 :
       self.laser flag = True # LaserScan 메시지
   else:
       self.laser flag = False # 유효하지 않은 메.
def sense(self): # 감지 함수
    pub_dist90_R_lsit = []
    pub dist90 L lsit = []
    pub dist45 R lsit = []
    pub_dist45_L_lsit = []
    pub_obs_R_list = []
    pub_obs_L_list = []
    pub_obs_C_list = []
```

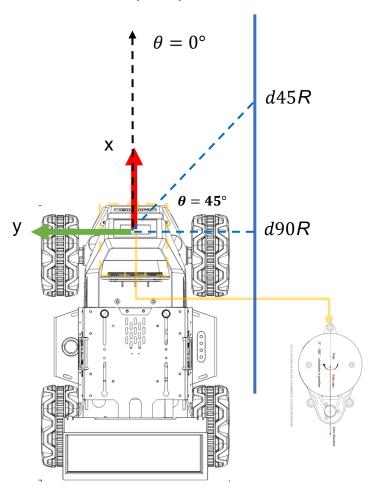
라이다를 통한 장애물 인식 실습

-. 장애물 인식(판단) 알고리즘



```
for i, n in enumerate(self.ranges):
   ## x, y 좌표로 변환
   x = n * math.cos(self.degrees[i] * math.pi / 180)
   y = n * math.sin(self.degrees[i] * math.pi / 180)
   # right obstacle detection
   if 0.15 < y < 0.45 and 0 < x < 0.5:
       pub_obs_R_list.append(n)
   # front obstacle detection
   if -0.15 < y < 0.15 and 0 < x < 0.5:
       pub_obs_C_list.append(n)
   # left obstacle detection
   if -0.45 < y < -0.15 and 0 < x < 0.5:
       pub_obs_L_list.append(n)
```

- 라이다를 통한 장애물 인식 실습
 - -. 장애물 인식(판단) 알고리즘

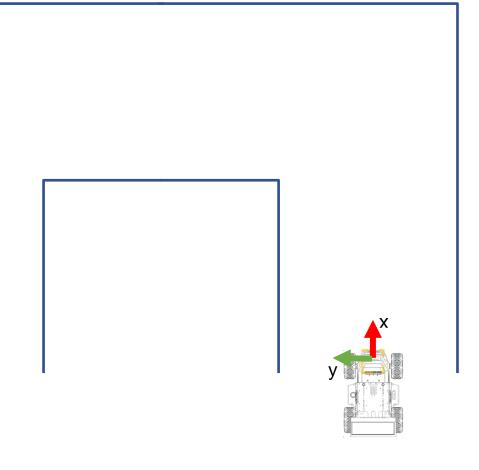


```
60 ~
                for i, n in enumerate(self.ranges):
61
                    ## x, y 좌표로 변환
                    x = n * math.cos(self.degrees[i] * math.pi / 180)
62
                    y = n * math.sin(self.degrees[i] * math.pi / 180)
63
                    # right obstacle detection
64
                    if 0.15 < y < 0.45 and 0 < x < 0.5:
65 ~
                        pub_obs_R_list.append(n)
66
                    # front obstacle detection
67
                    if -0.15 < y < 0.15 and 0 < x < 0.5:
68 ~
                        pub_obs_C_list.append(n)
69
                    # left obstacle detection
70
                    if -0.45 < y < -0.15 and 0 < x < 0.5:
71 ~
72
                        pub_obs_L_list.append(n)
                    # 오른쪽 90도 거리 저장
74
                    if 0 < n < 0.5 and -91 < self.degrees[i] < -89:
75
76
                        pub_dist90_R_lsit.append(n)
77
                    # 오른쪽 45도 거리 저장
                    if 0 < n < 0.5 and -46 < self.degrees[i] < <math>-44:
78
                        pub_dist45_R_lsit.append(n)
79
```

01 LIMO 원격접속 및 ROS 환경설정

- 라이다를 통한 터널 통과(wall follower)
 - -. Wall follower 알고리즘

Front	Left	Right	Action
off	off	off	go straight
off	on	on	following the right wall
off	off	on	following the right wall
off	on	off	following the left wall
on	on	off	Turn right
on	on	on	Stop or back
on	off	on	Turn left

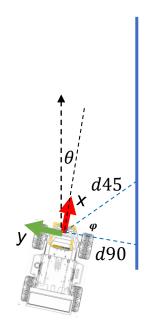


on LIMO 원격접속 및 ROS 환경설정

• 라이다를 통한 터널 통과(wall follower)

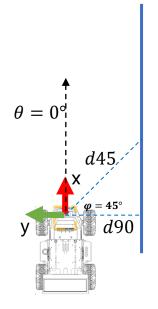
-. Wall follower 알고리즘

Front	Left	Right	Action
off	off	off	go straight
off	on	on	following the right wall
off	off	on	following the right wall
off	on	off	following the left wall
on	on	off	Turn right
on	on	on	Stop or back
on	off	on	Turn left



$$\cos \varphi = \frac{d45}{d90}$$
 $\theta = 0$ deg 이려면, $\varphi = 45$ deg

$$\varphi = a\cos\frac{d45}{d90}$$
 따라서, $e(t) = \left(\frac{\pi}{4} - \varphi\right) + (dist - d90)$

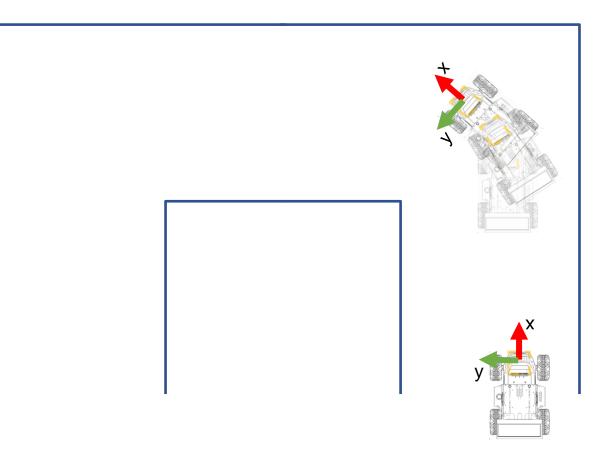


▶ desired 벽과의 거리

01 LIMO 원격접속 및 ROS 환경설정

- 라이다를 통한 터널 통과(wall follower)
 - -. Wall follower 알고리즘

Front	Left	Right	Action
off	off	off	go straight
off	on	on	following the right wall
off	off	on	following the right wall
off	on	off	following the left wall
on	on	off	Turn right
on	on	on	Stop or back
on	off	on	Turn left



- 라이다를 통한 터널 통과(wall follower)
 - -. Wall follower 알고리즘

Front	Left	Right	Action
off	off	off	go straight
off	on	on	following the right wall
off	off	on	following the right wall
off	on	off	following the left wall
on	on	off	Turn right
on	on	on	Stop or back
on	off	on	Turn left

```
64 V
65 V
66
67
68 ~
69
70
71
72 V
73
74
75 V
76
77
78
79
80
81
82
```

```
def control(self): # 제어 함수
    if self.obsC and self.obsR and self.obsL: # Stop
       speed = 0
       steer = 0
   elif not self.obsC and self.obsR: # following the right wall
       speed = 0.2
       theta = acos(self.dist45R / self.dist90R)*180/pi
       steer = (45-theta) + self.kp*(self.wall_dist - self.dist90R)
   elif self.obsC and self.obsR: # turn left
       speed = 0.1
       steer = 0.1
    else:
       print('조건 없음')
       speed = 0
       steer = 0
    self.cmd_msg.linear.x = speed
   self.cmd_msg.angular.z = steer
    self.pub.publish(self.cmd msg)
```