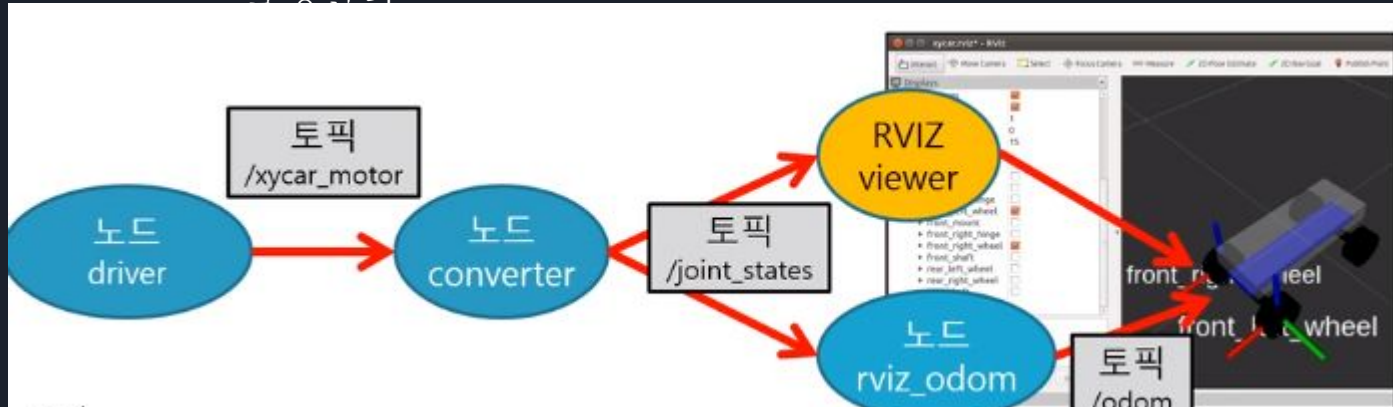


3D 자동차 제어 프로그래밍 과제

프로그래머스 자율주행 코스 1기 조정민

목표

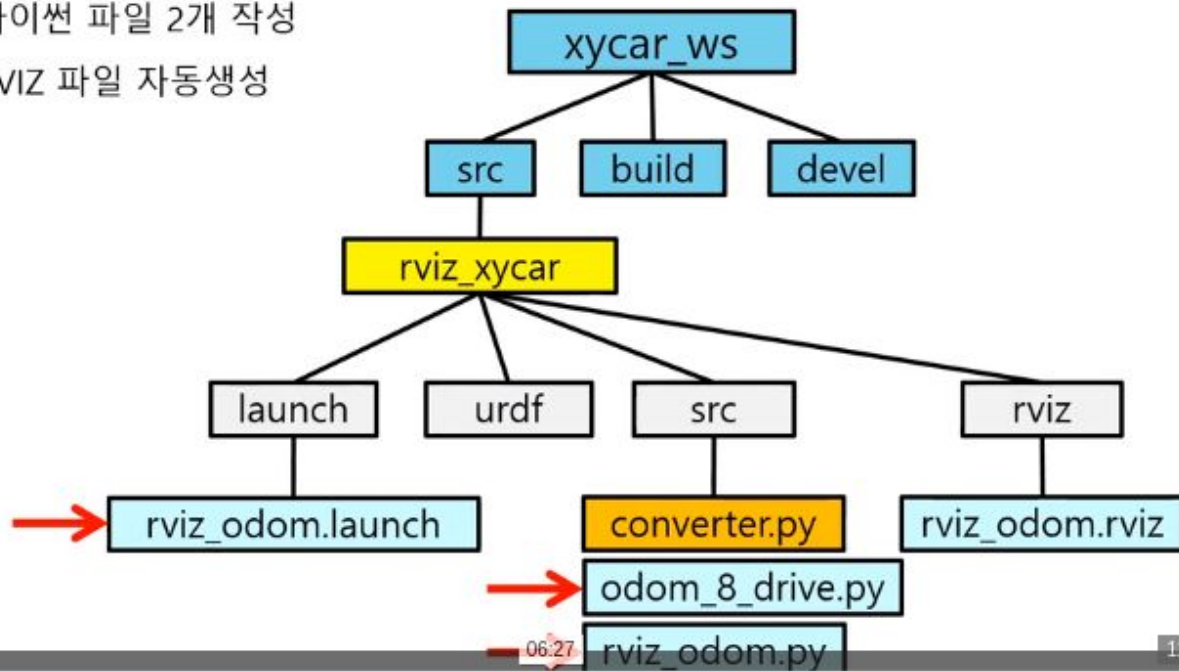
- RVIZ 가상공간에 있는 3D 자동차를 주행시켜보자
- 동작과정
 - 8자 주행 프로그램이 모터제어 메시지를 보내면 (/xycar_motor 토픽)
 - 그걸 변환 프로그램이 받아서 /joint_states 토픽을 만들어 발행하고
 - 그걸 또 오도메트리 프로그램이 받아서 변환해 /odom 토픽을 만들어 발행한다



파일 구성

- 기존 rviz_xycar 패키지 사용

- 런치 파일 1개 작성
- 파이썬 파일 2개 작성
- RVIZ 파일 자동생성



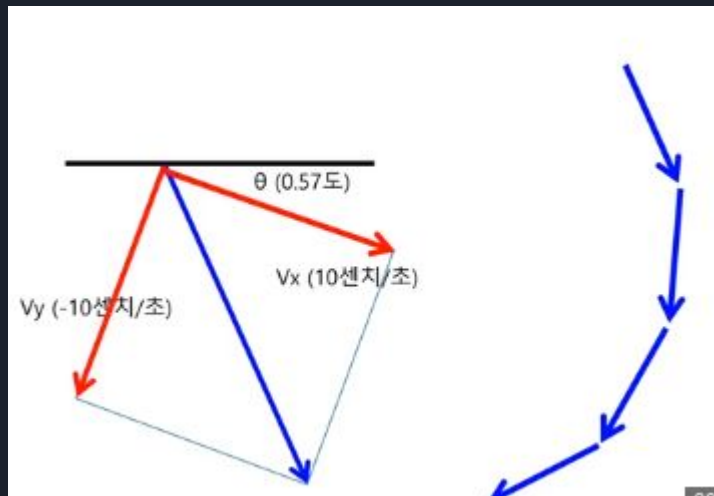


odom_8_drive.py

- 8차 주행을 하도록 운전하는 프로그램
- /xycar_motor 토픽 발행
- 이전 코드에서 시간쪽만 더 추가

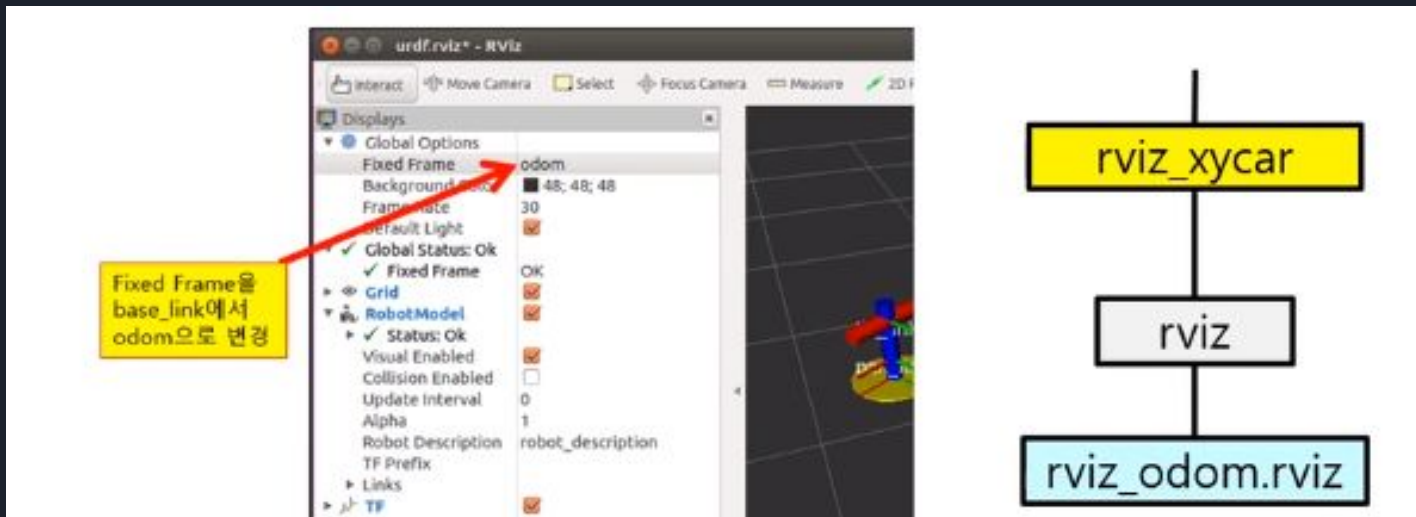
rviz_odom.py

- converter 노드가 보내는 /joint_states 토픽을 받아서 바퀴의 방향과 회전속도 정보를 획득하고
- 그 정보를 바탕으로 오도메트리 데이터를 만들어 /odom 토픽에 담아 발행한다.
- 바퀴 크기 원주 고려



rviz_odom.rviz(자동생성)

- RVIZ 뷰어 화면에 차량 3D 그림이 잘 표시되도록 설정하고
- RVIZ 끝낼때 설정내용을 잘 SAVE한다.
- 저장 위치는 .launch 파일에서 지정할 수 있다.



rviz_odom.launch

Launch 파일 - rviz_odom.launch

```
<launch>
  <param name="robot_description" textfile="$(find rviz_xycar)/urdf/xyicar_3d.urdf"/>
  <param name="use_gui" value="true"/>

  <!-- rviz display -->
  <node name="rviz_visualizer" pkg="rviz" type="rviz" required="true"
        args="-d $(find rviz_xycar)/rviz/rviz_odom.rviz"/>


  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher"/>
```

추가 → 노드 실행 : odom_8_drive.py

추가 → 노드 실행 : rviz_odom.py

추가 → 노드 실행 : converter.py

```
</launch>
```



실행 화면

```
$ roslaunch rviz_xycar rviz_odom.launch
```


1. odom_8_drive.py 작성

odom_8_drive.py에서
이전 파일 복사

- 첫번째 for문은
좌회전 또는 우회전
(10 or -10)
토픽을 보내고
- 두번째 for문은
직진 토픽 (0)을
보낸다.

```
10 rospy.init_node('auto_driver')
11
12 pub = rospy.Publisher('xycar_motor', xycar_motor, queue_size=1)
13
14 def motor_pub(angle, speed):
15     # global pub
16     motor_control = xycar_motor()
17     motor_control.header = Header()
18     motor_control.header.stamp = rospy.Time.now()
19     motor_control.angle = angle
20     motor_control.speed = speed
21
22     pub.publish(motor_control)
23
24 speed = 6
25 turn_angle = 10
26 straight_angle = 0
27 rate = rospy.Rate(10)
28
29 while not rospy.is_shutdown():
30
31     # 좌회전 또는 우회전
32     for _ in range(0,250):
33         motor_pub(turn_angle,speed)
34         rate.sleep()
35
36     for _ in range(0,75):
37         motor_pub(straight_angle,speed)
38         rate.sleep()
39
40     # 좌회전 방향이면 우회전으로, 우회전 방향이면 좌회전으로 변경
41     turn_angle *= -1
```

2. converter.py 수정

- `mt_data.speed`에 비례하게끔 바퀴 굴러가는 속도를 만든다.
- 바퀴 각도 (`mt_angle`) 과 바퀴가 구른 정도 (`a, b`), 차량 속도 (`speed`) 를 `JointState` 메시지에 저장하고 토픽을 보낸다.

```
8  a = -3.14
9  b = -3.14
10
11 def callback(mt_data):
12     global a
13     global b
14     wheel_msg = JointState()
15     wheel_msg.header = Header()
16     wheel_msg.name = ['front_right_hinge_joint', 'front_left_hinge_joint',
17                       'front_right_wheel_joint', 'front_left_wheel_joint',
18                       'rear_right_wheel_joint', 'rear_left_wheel_joint']
19     wheel_msg.velocity = [mt_data.speed]
20     wheel_msg.effort = []
21     wheel_msg.header.stamp = rospy.Time.now()
22
23     if a >= 3.14:
24         a = -3.14
25         b = -3.14
26     else:
27         # 0.01 라디안은 약 6도
28         a += 0.005 * mt_data.speed
29         b += 0.005 * mt_data.speed
30
31     # front_*_wheel_joint 만 회전 시키기 위한 것
32     wheel_msg.position = [mt_data.angle / 50, mt_data.angle / 50, a, b, 0, 0]
33     pub.publish(wheel_msg)
34
35 motor_control = xycar_motor()
36 rospy.init_node('converter')
37 sub = rospy.Subscriber('xycar_motor', xycar_motor, callback)
38 pub = rospy.Publisher('joint_states', JointState, queue_size=10)
39 rospy.spin()
```

3. rviz_odom.py - 1

- odom 토픽을 보내는 퍼블리셔와 joint_states 토픽을 받는 서브스크라이버를 생성하고 callback 함수를 지정한다.
- callback 함수에서 사용할 전역변수들을 지정한다.

```
67  rospy.init_node('rviz_odom')
68
69  odom_pub = rospy.Publisher("odom", Odometry, queue_size=50)
70  sub = rospy.Subscriber('joint_states', JointState, callback)
71  odom_broadcaster = tf.TransformBroadcaster()
72
73
74  x = 0.0
75  y = 0.0
76  th = 0.0
77  last_whl_ang = 0.0
78  last_time = rospy.Time.now()
79
80
81  rospy.spin()
```

3. rviz_odom.py - 2

- callback 함수에서 이전과의 시간차이 (dt), 각도차이 (delta_whl_ang) 등을 구한다.
- dt 동안의 이동거리는 바퀴의 delta_whl_ang 만큼의 호의 길이와 같다. 따라서 $2 * \text{delta_whl_ang} * 1.8$ (바퀴 반지름) 이 된다.
- dt 동안 이동한 현재 위치는 x좌표는 cos 값을 곱한 값과 y좌표는 sin 값을 곱한 값과 같다.

```
13 def callback(joint_data):
14     global x, y, th, last_whl_ang, last_time
15     current_time = rospy.Time.now()
16     curr_whl_ang = joint_data.position[2]
17
18     # when last_whl_ang >= 3.14 and curr_whl_ang == -3.14, delta_ang is sum of them.
19     delta_whl_ang = curr_whl_ang + last_whl_ang if (curr_whl_ang * last_whl_ang < 0) else curr_whl_ang - last_whl_ang
20
21     # distance = 2 * delta_angle * wheel's radius
22     vx = 2 * delta_whl_ang * 1.8
23     vy = 2 * delta_whl_ang * 1.8
24
25     dt = (current_time - last_time).to_sec()
26     delta_x = vx * cos(th) * dt
27     delta_y = vy * sin(th) * dt
28     delta_th = joint_data.position[0] * dt
29
30     x += delta_x
31     y += delta_y
32     th += delta_th
33
```

3. rviz_odom.py - 3

- 오일러 각도를 quaternion 값으로 변환한다.
- rviz 상에 위치시킬 곳을 지정한다.
- odom토픽을 생성하고 발행한다.
- last_time , angle을 최신 값으로 변경한다.

```
34 # since all odometry is 6DOF we'll need a quaternion created from yaw
35 # oiler value -> quaternion value
36 odom_quat = tf.transformations.quaternion_from_euler(0, 0, th)
37
38 # first, we'll publish the transform over tf
39 odom_broadcaster.sendTransform(
40     (x, y, 0.),
41     odom_quat,
42     current_time,
43     "base_link",
44     "odom"
45 )
46
47 # next, we'll publish the odometry message over ROS
48 odom = Odometry()
49 odom.header.stamp = current_time
50 odom.header.frame_id = "odom"
51
52 # set the position
53 odom.pose.pose = Pose(Point(x, y, 0.), Quaternion(*odom_quat))
54
55 # set the velocity
56 odom.child_frame_id = "base_link"
57 odom.twist.twist = Twist(Vector3(vx, vy, 0), Vector3(0, 0, joint_data.velocity[0] * dt))
58
59 # publish the message
60 odom_pub.publish(odom)
61
62 last_time = current_time
63 last_whl_ang = curr_whl_ang
```

실행화면

