ROS_4일차_보고서_20기_인턴_2025402040_컴퓨터정보공학부_노기문

-조이스틱으로좌표계이동하기-

목차

**듀얼쇼크4 연결**

ps home 버튼과 share 버튼을 동시에 눌러 블루투스와 연결한다.

## joy node란?

joy 노드는 ROS 2에서 조이스틱(게임패드) 장치를 표준화된 메시지로 변환해 주는 입력 드라이버다.

역할

Linux의 /dev/input/js* 장치에서 데이터를 읽는다.
읽은 데이터를 sensor_msgs/msg/Joy 타입으로 발행한다.
이 토픽을 다른 노드(예: teleop_twist_joy, 사용자 정의 퍼블리셔)가 받아 로봇 속도 명령이나 이벤트로 사용한다.

메시지 형식

```
sensor_msgs/msg/Joy
        std_msgs/Header header
        float32[] axes      # 아날로그 축 값. 범위 보통 -1.0 ~ 1.0
        int32[] buttons     # 디지털 버튼 값. 0 또는 1
```

예시 (ros2 topic echo /joy):

```
header:
  stamp:
    sec: 1758031130
    nanosec: 117799537
  frame_id: ''
axes: [0.0, 1.0, 0.0, -0.5]    # 왼쪽 스틱, 오른쪽 스틱 값
buttons: [0, 1, 0, 0, 1, 0]    # 버튼 눌림 여부
```

주요 파라미터

device : 사용할 조이스틱 장치 파일 (/dev/input/js0 기본).
deadzone : 작은 입력 무시 범위. 예: 0.05 → ±0.05는 0으로 처리.
autorepeat_rate : 같은 값이 유지될 때 반복 발행 속도(Hz).

coalesce_interval : 이벤트 묶음을 한번에 발행할 시간 간격.
dev_ff : 힘-피드백(force feedback) 장치 지원 여부.
실행 방법
ros2 run joy joy_node
혹은 launch 파일 내에서:

```
Node(
   package='joy',
   executable='joy_node',
   name='joy_node',
   output='screen',
   parameters=[{'deadzone': 0.05}]
)
```

활용 흐름
joy_node 실행 => /joy 토픽 발행.
다른 노드에서 /joy 구독 => 스틱 좌표/버튼값을 해석.
예: 스틱을 속도 명령(geometry_msgs/Twist)으로 매핑해서 /cmd_vel 퍼블리시.
즉, joy 노드는 하드웨어 의존 입력 => ROS 표준 메시지로 변환해 주는 최소 단위 드라이버다.

publisher.cpp

조이스틱 축값([-1,1])을 선속도·각속도에 선형 매핑해 /cmd_vel로 퍼블리시한다. 입력 잡음을 데드존으로 제거한다.

입출력

입력 토픽: /joy (sensor_msgs/msg/Joy)

axes[axis_linear_]: 전후 스틱 값

axes[axis_angular_]: 좌우 스틱 값

출력 토픽: /cmd_vel (geometry_msgs/msg/Twist)

linear.x = v [m/s]

angular.z = ωz [rad/s]

파라미터 의미

axis_linear 기본 1. 선속도에 쓸 축 인덱스.

axis_angular 기본 3. 각속도에 쓸 축 인덱스.

scale_linear 기본 1.0. 선속도 스케일 계수.

scale_angular 기본 1.0. 각속도 스케일 계수.

deadzone 기본 0.05. 축 절댓값이 이 값 미만이면 0으로 처리.

알고리즘 단계

/joy 콜백 수신 시 축 길이 검사

if axes.size() <= max(axis_linear, axis_angular) → return

데드존 함수 적용

듀얼쇼크 부호 보정 및 선형 매핑

듀얼쇼크 상(앞) = -1. 전진을 +로 만들기 위해 선속도에 부호 반전.

각속도는 오른쪽 입력을 +로 사용.

a_lin = msg.axes[axis_linear]

a_ang = msg.axes[axis_angular]

v = - dz(a_lin) * scale_linear

ωz = dz(a_ang) * scale_angular

Twist 메시지 생성·퍼블리시

t.linear.x = v

t.angular.z = ωz

publish(t)

반복. 콜백 기반이므로 /joy가 들어올 때만 출력이 갱신됨.

좌표계 의미

ROS 표준: 전진 +x, 좌회전 +yaw(+z).

위 매핑으로 전진 스틱↑가 linear.x > 0, 스틱→가 angular.z > 0이 됨.

## subscriebr.cpp

입출력

입력: /cmd_vel (geometry_msgs/Twist)

v = linear.x [m/s], ω = angular.z [rad/s]

출력 1: TF map → odom (TransformBroadcaster)

출력 2: /odom (nav_msgs/Odometry)

상태와 시간

상태벡터: p = [x, y, θ]^T

내부 상태: x_, y_, th_

제어: v_, wz_

시간 간격: dt = (now - last).seconds()

타이머 20 ms(50 Hz). /cmd_vel은 콜백에서 최신값으로 보관. 샘플 사이에 Zero-Order Hold 가정.

이 아래부터 수치적 해석이 들어가는데 아직 이해를 못하여 gpt의 도움을 받았습니다.

### 1. 상태(로봇 위치·각도)

- $x, y$: 로봇의 평면 위치
- $\theta$: 로봇의 바라보는 각도(yaw)

### 2. 입력(조이스틱에서 온 속도)

- $v$: 직진 속도 (m/s) → `linear.x`
- $\omega$: 회전 속도 (rad/s) → `angular.z`

### 3. 연속적인 운동 방정식

로봇이 이렇게 움직인다고 가정한다:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega$$

- $\dot{x}, \dot{y}, \dot{\theta}$는 시간에 따른 변화율이다.
- 즉, 로봇이 앞으로 갈 때는 $x, y$가 $\theta$ 방향으로 바뀌고, 회전할 때는 $\theta$가 $\omega$만큼 변한다.

### 4. 컴퓨터 안에서 적분 (이산 시간)

코드는 연속값을 직접 다룰 수 없으니 **작은 시간 간격 dt마다 값 업데이트**한다.
전진 오일러 방법(Euler integration):

$$\theta \leftarrow \theta + \omega \cdot dt$$

$$x \leftarrow x + v\cos\theta \cdot dt$$

$$y \leftarrow y + v\sin\theta \cdot dt$$

즉, "각속도 × 시간"만큼 각도를 늘리고, "선속도 × 시간"만큼 좌표를 이동시킨다.

제가 이해한 바로는 직진과 회전 속도를 입력하여 전 속도와의 차이를 미,적분 한뒤 그만큼 이동하는것이라고 이해했습니다. 즉, 조이스틱으로 들어온 속도(v, 각속도)를 작은 시간마다 더해서 누적하면서 로봇의 위치(x,y)와 각도(θ)를 계산하는 공식입니다.

result screen
영상 참조 바람.