ROS2 통신 – Launch

경희대학교 기계공학과
로봇 제어 지능 연구실
김상현 교수

1 Launch File

ROS2 Launch File

Launch

Launch의 장점

1) 여러 노드를 동시에 실행 할 수 있다.
2) Argument나 parameter 등을 쉽게 설정할 수 있게 되어 있어 관리하기 쉽다.
3) 코드를 쉽게 재사용 할 수 있다.

Ros2 run vs Launch File

Launch File

1) 시작 파일의 모든 노드는 거의 동시에 시작됩니다. 결과적으로 노드가 자체적으로 초기화되는 순서를 확신할 수 없습니다. 잘 작성된 ROS 노드는 자신과 형제 노드가 시작되는 순서에 신경 쓰지 않습니다.
2) 기본적으로 시작된 노드의 표준 출력은 터미널이 아니라 로그 파일(~/.ros/log/〈ros2_runtime〉/node/〈unique_n.log)입니다.
   〈ros2_runtime〉:setup.bash를 통해 활성화된 ID
   〈unique_node_id〉:실행중인 각 노드에 할당된 고유 id
3) 활성 ros2 launch를 종료하려면 Ctrl-C를 사용하세요.
   이 신호는 실행 시 각 활성 노드를 정상적으로 종료하려고 시도합니다.

ROS2 Launch File

Launch File의 구성

이름 및 위치

1) 각 실행 파일은 특정 패키지와 연결되어야 합니다.
2) 일반적인 명명 방식은 .py로 끝나는 시작 파일 이름을 지정하는 것입니다.
3) 패키지 디렉토리에 직접 저장되거나 일반적으로 launch라는 하위 디렉토리에 저장됩니다.

파일 구조

1) Root

```
def generate_launch_description():
        return LaunchDescription([])
```

LaunchDesrciption 안에서 Node,Action,Service가 실행 가능

2) Node

```
Node(
        package="패키지 이름",
        executable="실행파일",
        name="실행 노드",
        parameter=[{'name': 'value'}]
        remapping=[('old topic name','new topic name')])
```

3) Arg

```
DeclareLaunchArgument('변수이름',default_value='기본 변수',description='변수 설명')
```

ROS2 Launch File

- (실습 1) randvelturtlesim.launch.py 파일을 만들어 터틀봇Sim을 실행하는 노드를 만듭니다.

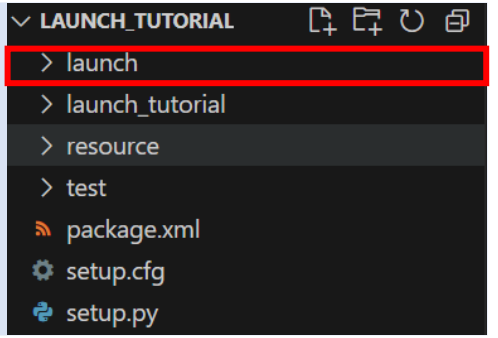
Package 만들기

$ ros2 pkg create launch_tutorial --build-type ament_python --dependencies topic_tutorial_py rclpy turtlesim

Launch 폴더 만들기

$ cd launch_tutorial

$ mkdir launch # 혹은 vscode로 launch 폴더 만들기



package.xml 설정

$ <exec_depend>ros2launch</exec_depend> 추가

setup.py 설정

```
import os
from glob import glob
 ...
data_files=[
      ('share/ament_index/resource_index/packages',
         ['resource/' + package_name]),
      ('share/' + package_name, ['package.xml']),
      (os.path.join('share', package_name, 'launch'), glob('launch/*.py'))
  ],
```

Launch 폴더 내부의 .py파일을 전부 설치

ROS2 Launch File

- (실습 1) randvelturtlesim.launch.py 파일을 만들어 터틀봇Sim을 실행하는 노드를 만듭니다.

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtle1'
        ),
    ])
```

turtlesim 패키지의 turtlesim_node 실행

빌드 후 실행 (ros2_ws에서)

$ colcon build

$ ros2 launch launch_tutorial randvelturtlesim.launch.py

ROS2 Launch File

- (실습 2-3) randvelturtlesim.launch.py 파일을 수정하여, pubvel과 subpose 노드를 동시에 실행할 수 있도록 합니다.

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtle1'
        ),
        Node(
            package='topic_tutorial_py',
            executable='pub_vel_node',
            name='vel_publisher',
            output='screen'
        ),
        Node(
            package='topic_tutorial_py',
            executable='sub_pose_node',
            name='pose_subscriber',
            output='screen'
        )
    ])
```

turtlesim 패키지의 turtlesim_node 실행

topic_tutorial_py 패키지의 pub_vel_node 실행

topic_tutorial_py 패키지의 sub_pose_node 실행

빌드 후 다시 실행

$ ros2 launch launch_tutorial randvelturtlesim.launch.py

ROS2 Launch File

- (실습 4) turtleflag.launch.py 파일을 만들어, 터틀봇 Sim을 만들고 임의의 argument를 1로 설정하면 원격 조작이 활성화되도록 teleop 노드의 옵션을 설정합니다. 0이 들어왔을 땐 pubvel과 subpose를 실행하도록 런치파일을 만들어 봅니다.

우선 설치 사항

```
$ sudo apt install xterm
```

핵심 문법 (argument 만들기)

```
DeclareLaunchArgument(
        'use_teleop',
        default_value='1',
        description='Flag to indicate whether to use teleop or not'
    ),
```

Argument 설정

핵심 문법 (argument 사용여부 결정)

```
condition=IfCondition(LaunchConfiguration('use_teleop'))

condition=UnlessCondition(LaunchConfiguration('use_teleop'))
```

Argument 사용 여부 결정

ROS2 Launch File

- (실습 4) turtleflag.launch.py 파일을 만들어, 터틀봇 Sim을 만들고 임의의 argument를 1로 설정하면 원격 조작이 활성화되도록 teleop 노드의 옵션을 설정합니다. 0이 들어왔을 땐 pubvel과 subpose를 실행하도록 런치파일을 만들어 봅니다.

```python
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument
from launch.conditions import IfCondition, UnlessCondition
from launch.substitutions import LaunchConfiguration

def generate_launch_description():
    return LaunchDescription([
        DeclareLaunchArgument(
            'use_teleop',
            default_value='1',
            description='Flag to indicate whether to use teleop or not'
        ),

        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim_node',
            output='screen'
        ),

        Node(
            package='turtlesim',
            executable='turtle_teleop_key',
            name='teleop_key',
            output='screen',
            prefix='xterm -e',
            condition=IfCondition(LaunchConfiguration('use_teleop'))
        ),

        Node(
            package='topic_tutorial_py',
            executable='pub_vel_node',
            name='vel_publisher',
            output='screen',
            condition=UnlessCondition(LaunchConfiguration('use_teleop'))
        ),

        Node(
            package='topic_tutorial_py',
            executable='sub_pose_node',
            name='pose_subscriber',
            output='screen',
            condition=UnlessCondition(LaunchConfiguration('use_teleop'))
        ),
    ])
```

Condition을 통해 argument에 따라 켜지는 노드가 바뀜

빌드 후 다시 실행 (두 가지 실행 결과를 비교해보세요)

$ ros2 launch launch_tutorial turtleflag.launch.py or $ ros2 launch launch_tutorial turtleflag.launch.py use_teleop:=0

ROS2 Launch File

- 추가 기능: 네임스페이스(namespace)를 활용한 다양한 Node 구성

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim',
            namespace='sim1',
            output='screen',
            respawn=True,
        ),
        Node(
            package='turtlesim',
            executable='turtle_teleop_key',
            name='teleop_key',
            namespace='sim1',
            output='screen',
            prefix='xterm -e',
        ),
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim',
            namespace='sim2',
            output='screen',
            respawn=True,
        ),
        Node(
            package='turtlesim',
            executable='turtle_teleop_key',
            name='teleop_key',
            namespace='sim2',
            output='screen',
            prefix='xterm -e',
        ),
    ])
```

- (실습 5) 왼쪽의 노드들을 실행할 수 있도록 런치파일을 구성하세요. (namespace.launch.py)
  그 뒤, rqt_graph를 활용하여 만든 launch 파일을 실행시켰을 때 node 구성을 확인해보세요.

Terminal 실행명령어

$ ros2 launch launch_tutorial namespace.launch.py

- turtle1/cmd_vel ->/sim1/turtle1/cmd_vel
  /sim2/turtle1/cmd_vel

- turtle1/pose ->/sim1/turtle1/pose
  /sim2/turtle1/pose

ros2 topic list로 확인

ROS2 Launch File

- 추가 기능: Remapping(ns)을 활용한 기존 Topic들을 활용

Remapping이란?

1) Node에 사용되는 Topic 이름들을 코드의 수정없이, Launch에서 수정할 수 있는 기능

```
Node(
        package='turtlesim',
        executable='turtlesim_node'
        name='turtlesim',
        remapping[('/turtle1/pose','tim')]
```

turtle1/pose로 퍼블리시 되는 토픽을 "tim"이라는 이름으로 재매핑

- 실습 6: Tele-op에서 발생하는 속도 (cmd_vel)을 읽어, 반대 방향으로 동작하게 코드를 작성

Terminal 실행명령어

```
ros2 launch launch_tutorial reverse.launch.py
```

reverse.launch.py에서

```
Node(
        package='turtlesim',
        executable='turtlesim_node'
        name='turtlesim',
        ...
        remapping[('/turtle1/cmd_vel','/turtle1/cmd_vel_reversed')]
```

Remapping을 통해 속도를 거꾸로 전달한다.

Launch 파일을 실행하면?
-tuetlesim_node실행-> cmd_vel대신 cmd_vel_reverse로 속도 구독
-reverse_node실행->teleop_key의 cmd_vel을 구독
-부호를 붙인 후 cmd_vel_reverse로 발간
reverse node 구성은 Lecture 4참고

03 ROS2 Parameter

ROS2 Parameter

Ros2 Parameter를 왜 사용하는가?

C++등에서는 자주 바꾸는 변수를 선언하면, 그때 그때 마다 빌드를 하여 활용하여야 함.

- ROS2에는 노드에서 쿼리할 값 모음을 추적하는 파라미터 서버가 있으며, 기본적으로 시간이 지나도 (많이) 변경되지 않는 구성 정보를 저장

실험 및 테스트가 용이

- 다양한 매개변수 값을 빠르게 변경하여 테스트를 진행할 수 있다.
- 매개변수를 분리함으로써 코드를 깔끔하게 유지하고 쉽게 변경할 수 있다.

- 현재 저장된 파라미터를 확인하고 싶을 때,

```
$ ros2 param list
```

- 현재 저장된 파라미터의 값을 확인하고 싶을 때,

```
$ ros2 param get parameter_name
```

- 현재 저장된 파라미터의 값을 세팅하고 싶을 때,

```
$ ros2 param set parameter_name value
```

ROS Parameter

- (실습 7) cmd_vel을 몇 Hz마다 발간할 지, 파라미터로 결정하여 줍시다.

  - Node파일

핵심 문법 (parameter 선언)

self.declare_parameter("parameter_name", default_value) ← parameter 이름 설정 및 default value 지정

핵심 문법 (parameter 가져오기)

self.get_parameter("parameter_name").get_parameter_value().<type>_value

Parameter을 가져오고 type에 맞춰 값을 가져오기

  - Launch 파일

핵심 문법

```
Node(
        package='launch_tutorial',
        executable='parameter_node',
        name='random_publish_velocity',
        parameters=[{'cmd_vel_rate': 5.0}],
        output='screen',
    ),
```

← Parameter 값 지정

감사합니다