반려로봇-2023

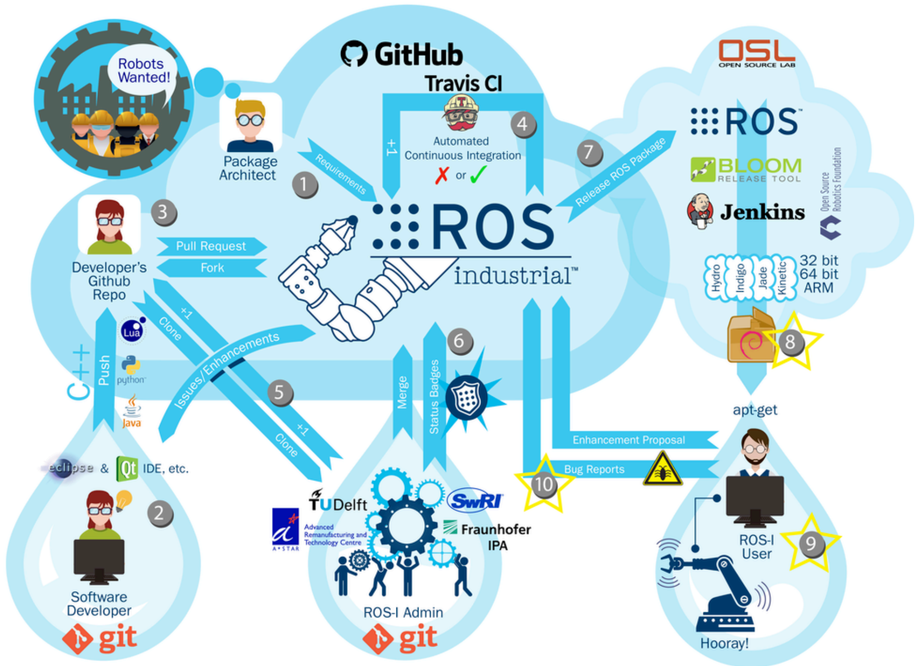
ROS2 소개

2023.07.15

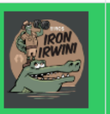
1. ROS2

ROS2 Version

- **참고 동영상**
  - https://www.youtube.com/watch?v=X9uYIumhU8E&list=PLieE0qnqO2kTNCznjLX_AaXe2hNJ-IpVQ
- **ROS 페이지**
  - https://docs.ros.org/en/humble/index.html
  - https://micro.ros.org/
- **버전 정보**
  - https://docs.ros.org/en/humble/Releases.html



참조: ROS Industrial

Distro	Release date	Logo	EOL date
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		June 20th, 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018
beta3	September 13th, 2017		December 2017
beta2	July 5th, 2017		September 2017
beta1	December 19th, 2016		Jul 2017
alpha1 - alpha8	August 31th, 2015		December 2016

- **디자인 컨셉**
  - ROS 2 Design

- **ROS1 vs ROS2**
  - No ROS Master (~~roscore~~)
  - No TCP/UDP, use DDS
  - Security & Performance Improve

- **DDS(Data Distribution Service)**
  - 실시간 데이터 분배 미들웨어
  - https://ai-sinq.tistory.com/entry/ROS2%EC%99%80-DDS%EB%9E%80

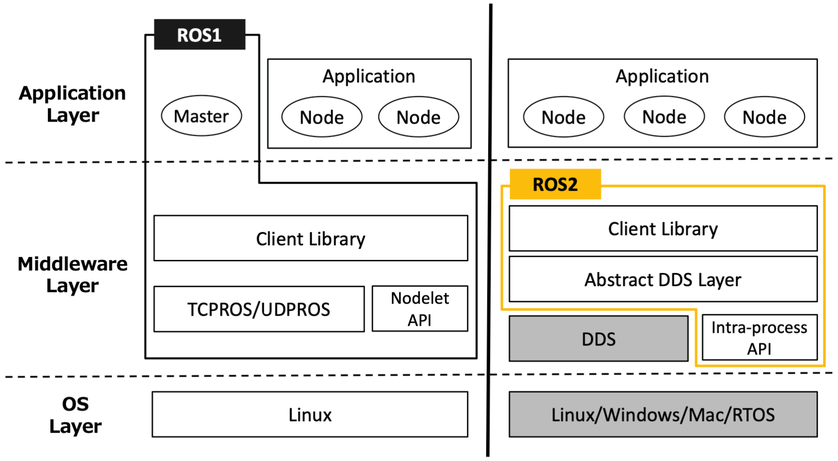


Figure 1: ROS1/ROS2 architecture.

참조: Exploring the Performance of ROS 2

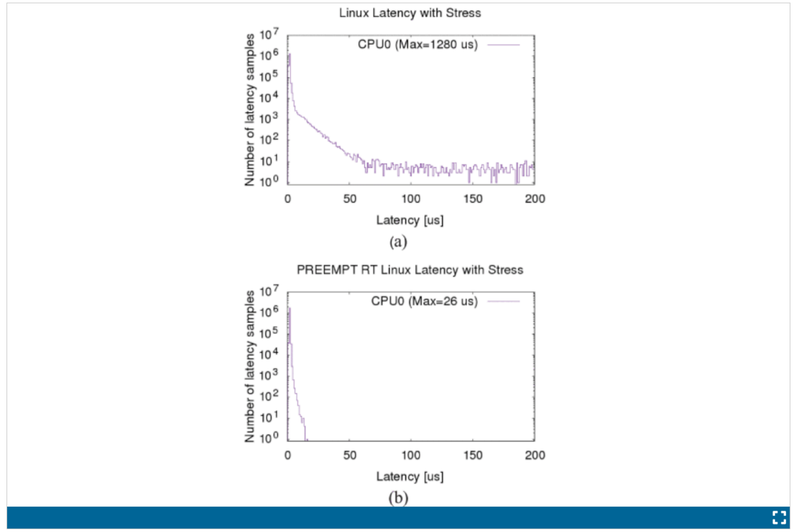


FIGURE 3.
Scheduling latency in a stressed environment. (a) ROS 1.0. (b) ROS 2.0.

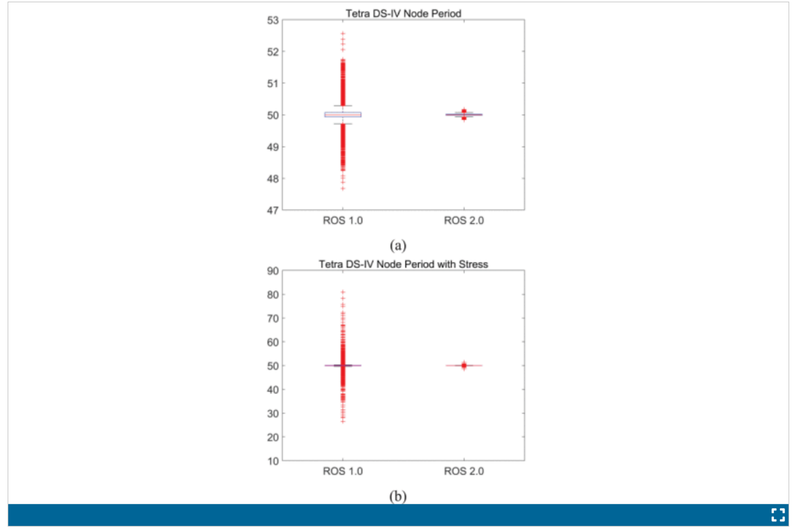


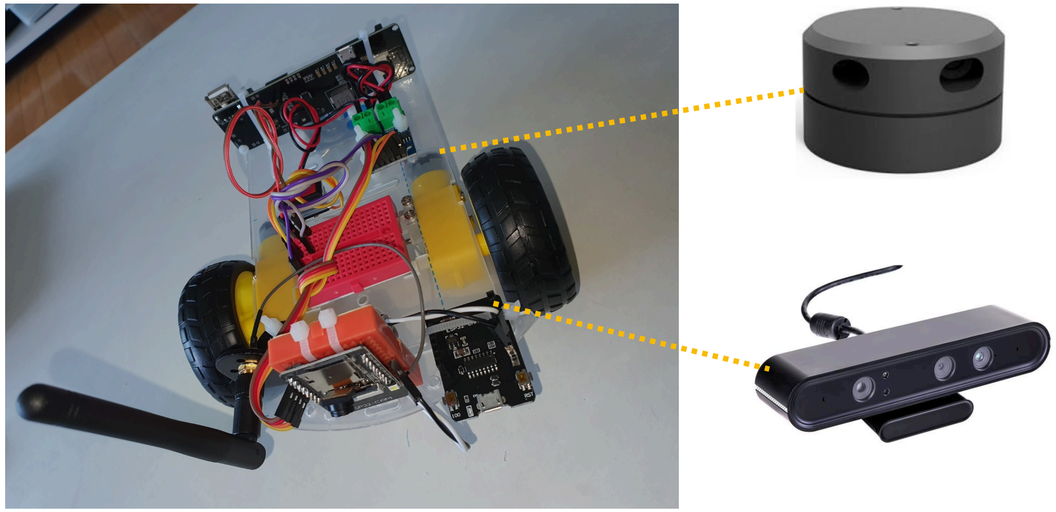
FIGURE 18.
Periodicity of the Tetra DS-IV node. (a) Idle environment (b) unstable environment with stress.

참조: Real-Time Characteristics of ROS 2.0

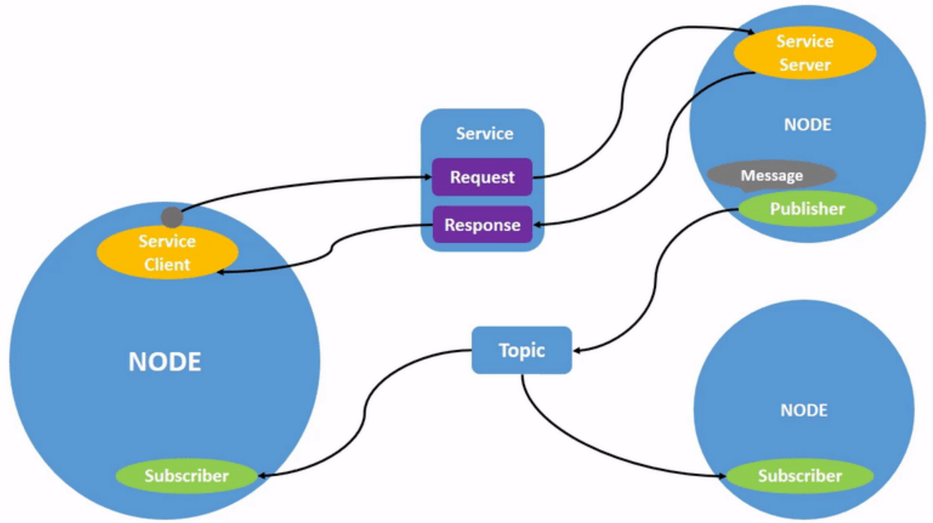
참고(로봇 메뉴얼): TETRA-DS IV Operation Manual

- **Roadbalance.com 김수영 대표**
  - https://www.youtube.com/watch?v=X9uYIumhU8E&list=PLieE0qnqO2kTNCznjLX_AaXe2hNJ-IpVQ
- **동아대 기계공학과 신승태 교수님**
  - https://www.youtube.com/watch?v=fbiT445DE9s&list=PL12w7vYWefUwSg5frw_tIHVp9XRwS8Ipt
- **ROS2 Tutorial**
  - https://docs.ros.org/en/humble/Tutorials.html

- **Node란?**
  - Each node in ROS should be responsible for a single, modular purpose, e.g. controlling the wheel motors or publishing the sensor data from a laser range-finder.
  - Each node can send and receive data from other nodes via topics, services, actions, or parameters



```
$ ros2 run turtlesim turtlesim_node
$ ros2 run turtlesim turtle_teleop_key
$ ros2 node list
$ ros2 node info /turtlesim
$ rqt_graph
```



참조: [Understanding ROS 2 nodes]

- **Package란?**
  - 특정 기능을 위한 node들을 그룹화 하는 단위

```
$ ros2 pkg create --build-type ament_cmake  <패키지이름>
$ ros2 pkg create --build-type ament_python <패키지이름>
$ mkdir -p ~/Workspace/ros_ws/src
$ cd ~/Workspace/ros_ws/src
$ ros2 pkg create --build-type ament_python  ros_tutorial
```

ROBOTIS-Will lds-02 support ··· ✓ 8237b79 on Feb 8, 2022 **770** commits

.github/workflows	update ros-tooling/setup-ros version	2 years ago
turtlebot3	lds-02 support	last year
turtlebot3_bringup	lds-02 support	last year
turtlebot3_cartographer	lds-02 support	last year
turtlebot3_description	lds-02 support	last year
turtlebot3_example	lds-02 support	last year
turtlebot3_navigation2	lds-02 support	last year
turtlebot3_node	lds-02 support	last year
turtlebot3_teleop	lds-02 support	last year
.gitignore	Added tb3_sbc_settings files.	4 years ago
CONTRIBUTING.md	modified ament_copyright	3 years ago
ISSUE_TEMPLATE.md	update	2 years ago
LICENSE	modified ament_copyright	3 years ago
README.md	update	2 years ago
turtlebot3.repos	rectify repos	3 years ago
turtlebot3_ci.repos	rectify repos	3 years ago

참조: turtlebot3

- **Launch란?**
  - Use a command line tool to launch multiple nodes at once.
  - 로봇의 실행 및 테스트를 위한 기능을 달성하기 위해 필요한 여러 노드들을 한꺼번에 실행시키기 위한 scripts

setup.py

```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```

Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/launch/turtlesim.launch.py

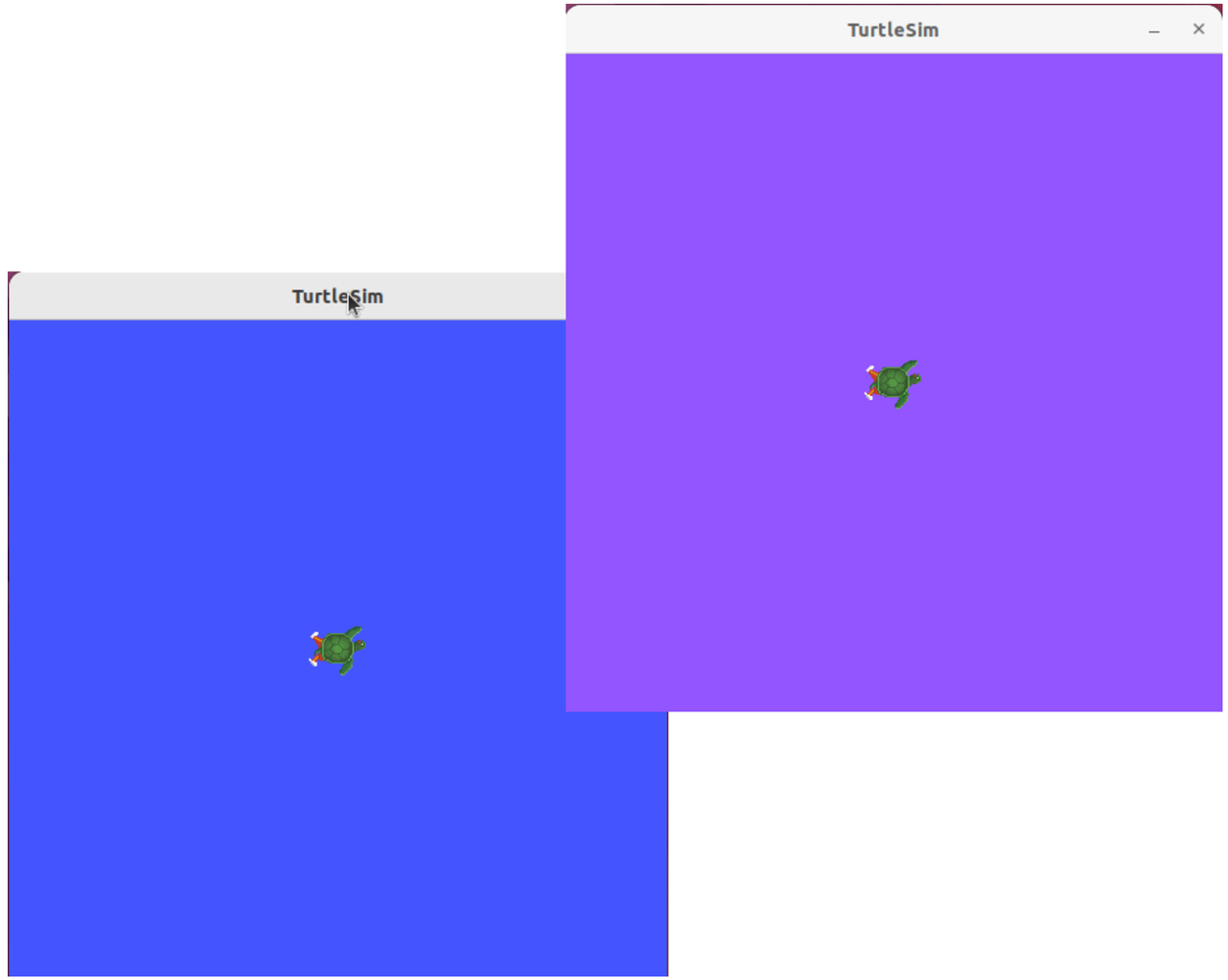
```
$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 run turtlesim turtle_teleop_key
```

turtlesim.launch.py

```python
#!/usr/bin/env python3

from launch import LaunchDescription
from launch.actions import ExecuteProcess
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription(
        [
            Node(
                package='turtlesim',
                executable='turtlesim_node',
                parameters=[],
                arguments=[],
                output='screen'
            ),
            ExecuteProcess(
                cmd=["ros2", "run", "rqt_graph", "rqt_graph"],
                output="screen",
            )
        ]
    )
```

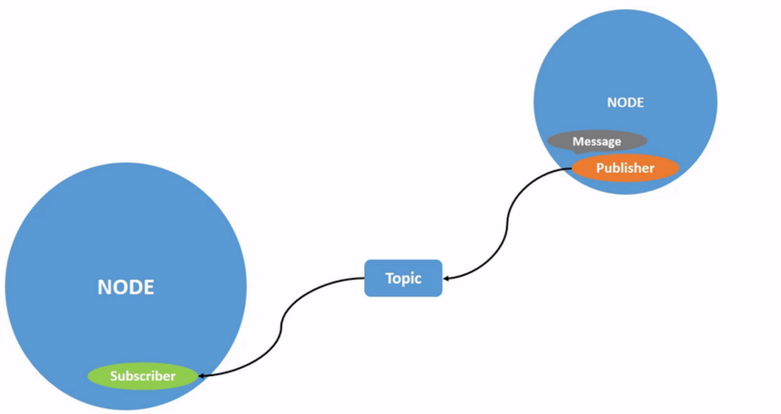
- **Parameter란?**
  - A parameter is a configuration value of a node.
  - You can think of parameters as node settings.
  - A node can store parameters as integers, floats, booleans, strings, and lists.
  - In ROS 2, each node maintains its own parameters.

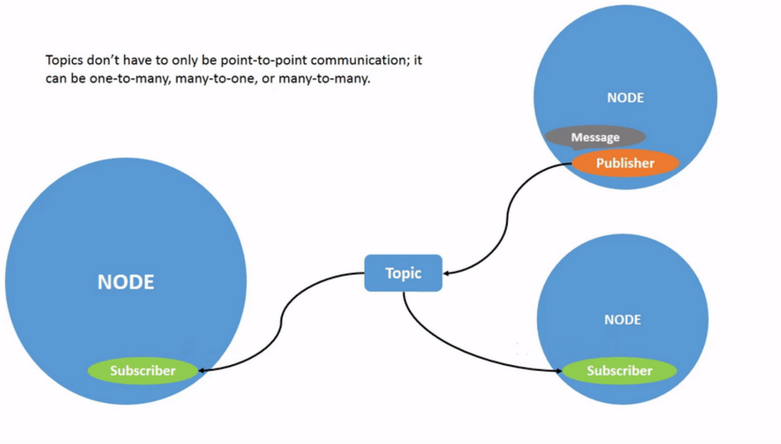
```
$ ros2 run turtlesim turtlesim_node
$ ros2 run turtlesim turtle_teleop_key
$ ros2 param list

$ ros2 param get <node_name> <parameter_name>
$ ros2 param get /turtlesim background_r
$ ros2 param set /turtlesim background_r 150
$ ros2 param get /turtlesim background_r

$ ros2 param dump /turtlesim > turtlesim.yaml
Edit turtlesim.yaml
$ ros2 param load /turtlesim turtlesim.yaml

Edit turtlesim.yaml
$ ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim.yaml
```



2. Topic, Service, Action

- **Topic란?**
  - ROS 2 breaks complex systems down into many modular nodes.
  - Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages.
  - A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.
  - Publisher node가 어떤 정보를 DDS를 Topic을 통해서 방송하고, 해당 정보를 필요로 하는 여러 Subscriber node(s)가 이를 수신해서 필요한 기능을 수행
  - N:N 통신이고 수신 여부에 대해서 확인하지 않음





참조: Understanding topics

```
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

```
$ ros2 topic list
$ ros2 topic info /turtle1/cmd_vel
$ ros2 topic find geometry_msgs/msg/Twist
$ ros2 interface show geometry_msgs/msg/Twist

Edit src/ros_tutorial/package.xml
Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/ros_tutorial/turtlesim_circle.py

$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 run ros_tutorial turtlesim_circle
$ ros2 topic echo /turtle1/cmd_vel
```

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.
<package format="3">
  <name>ros_tutorial</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="cchyun@todo.todo">cchyun</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>geometry_msgs</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

package.xml

```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'turtlesim_circle = ros_tutorial.turtlesim_circle:main',
        ],
    },
)
```

setup.py

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist, Vector3


class TurtlesimCirclePublisher(Node):
    def __init__(self):
        super().__init__('turtlesim_circle')

        self.publisher = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)  # 10: quesize
        self.timer = self.create_timer(0.5, self.timer_callback)

    def timer_callback(self):
        msg = Twist()
        msg.linear = Vector3(x=2.0, y=.0, z=.0)
        msg.angular = Vector3(x=.0, y=.0, z=1.8)
        self.publisher.publish(msg)


def main(args=None):
    rclpy.init(args=args)

    publisher = TurtlesimCirclePublisher()
    rclpy.spin(publisher)  # blocked until ros2 shutdown

    publisher.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

turtlesim_circle.py

Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/ros_tutorial/turtlesim_echo.py

$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 run ros_tutorial turtlesim_circle
$ ros2 run ros_tutorial turtlesim_echo

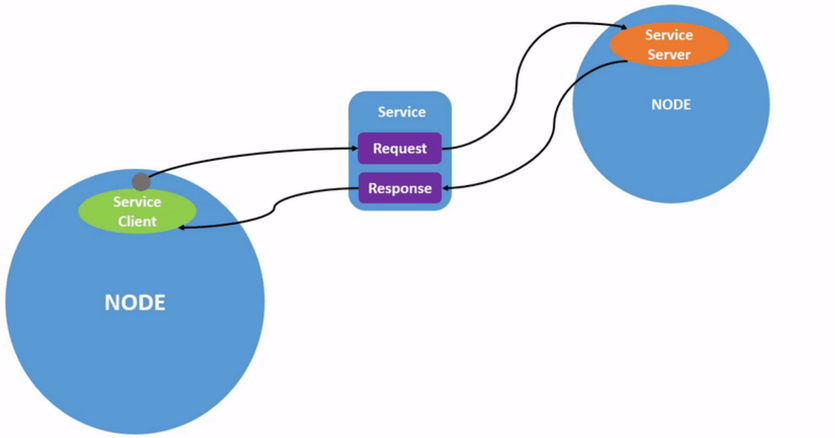
```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist, Vector3


class TurtlesimCircleSubscriber(Node):
    def __init__(self):
        super().__init__('turtlesim_echo')

        self.subscriber = self.create_subscription(Twist, '/turtle1/cmd_vel',
                                                   self.topic_callback, 10)  # 10: quesize

    def topic_callback(self, msg):
        self.get_logger().info(f'recv mes: {msg}')


def main(args=None):
    rclpy.init(args=args)

    subscriber = TurtlesimCircleSubscriber()
    rclpy.spin(subscriber)  # blocked until ros2 shutdown

    subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

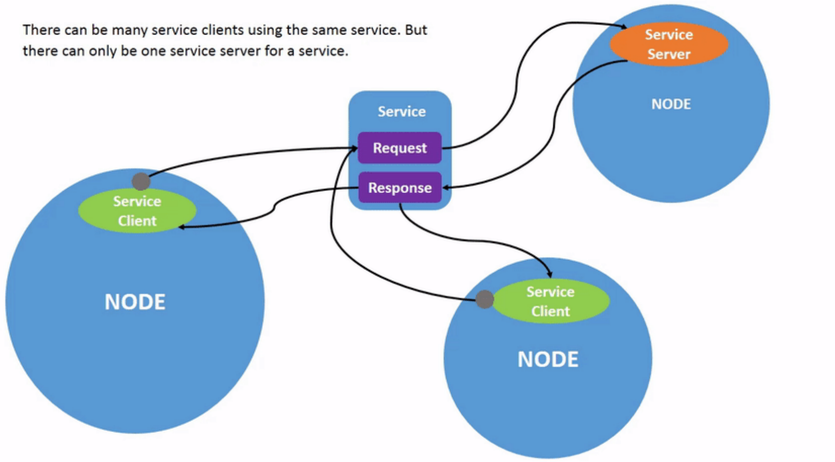
turtlesim_echo.py

```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'turtlesim_circle = ros_tutorial.turtlesim_circle:main',
            'turtlesim_echo = ros_tutorial.turtlesim_echo:main',
        ],
    },
)
```

setup.py

- **Service란?**
  - Services are another method of communication for nodes in the ROS graph.
  - Services are based on a call-and-response model versus the publisher-subscriber model of topics.
  - While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client.
  - 1:1 통신, 명령을 보내고 처리 결과를 수신하는 통신 방식





참조: Understanding services

```
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 service call /turtle1/teleport_absolute turtlesim/srv/TeleportAbsolute "{x: 5.5, y: 5.5, theta: 0.0}"
```

참고: Topics vs Services vs Actionlib...

```
$ ros2 service list
$ ros2 service type /turtle1/teleport_absolute
$ ros2 service find turtlesim/srv/TeleportAbsolute
$ ros2 interface show turtlesim/srv/TeleportAbsolute

Edit src/ros_tutorial/package.xml
Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/ros_tutorial/turtlesim_abs_client.py

$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 run ros_tutorial turtlesim_abs_client
```

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_form
<package format="3">
  <name>ros_tutorial</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="cchyun@todo.todo">cchyun</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>geometry_msgs</depend>
  <depend>turtlesim</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

package.xml

```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*.launch
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'turtlesim_circle = ros_tutorial.turtlesim_circle:main',
            'turtlesim_echo = ros_tutorial.turtlesim_echo:main',
            'turtlesim_abs_client = ros_tutorial.turtlesim_abs_client:main',
        ],
    },
)
```

setup.py

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from turtlesim.srv import TeleportAbsolute


class TurtlesimAbsoluteClient(Node):
    def __init__(self):
        super().__init__('turtlesim_abs_client')

        self.client = self.create_client(TeleportAbsolute, '/turtle1/teleport_absolute')

        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting again...')

        self.req = TeleportAbsolute.Request()

    def send_request(self, x, y, theta):
        self.req.x = x
        self.req.y = y
        self.req.theta = theta
        self.future = self.client.call_async(self.req)
        rclpy.spin_until_future_complete(self, self.future)
        return self.future.result()


def main(args=None):
    rclpy.init(args=args)

    client = TurtlesimAbsoluteClient()
    client.send_request(1.0, 1.0, 1.0)

    client.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

turtlesim_abs_client.py

Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/ros_tutorial/turtlesim_abs_server.py

$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 run ros_tutorial turtlesim_abs_server
$ ros2 run ros_tutorial turtlesim_abs_client

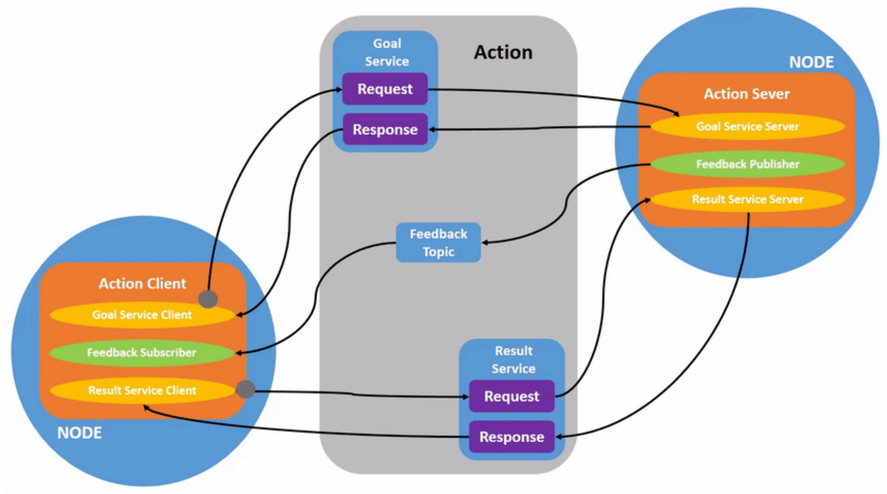
```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'turtlesim_circle = ros_tutorial.turtlesim_circle:main',
            'turtlesim_echo = ros_tutorial.turtlesim_echo:main',
            'turtlesim_abs_client = ros_tutorial.turtlesim_abs_client:main',
            'turtlesim_abs_server = ros_tutorial.turtlesim_abs_server:main',
        ],
    },
)
```

setup.py

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from turtlesim.srv import TeleportAbsolute


class TurtlesimAbsoluteServer(Node):
    def __init__(self):
        super().__init__('turtlesim_abs_server')

        self.client = self.create_service(TeleportAbsolute,
                                          '/turtle1/teleport_absolute',
                                          self.service_callback)

    def service_callback(self, request, response):
        print('request:', request)
        print('response:', response)
        return response


def main(args=None):
    rclpy.init(args=args)

    service = TurtlesimAbsoluteServer()
    rclpy.spin(service)

    service.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

turtlesim_abs_server.py

- **Action이란?**
  - Actions are one of the communication types in ROS 2 and are intended for long running tasks. They consist of three parts: a goal, feedback, and a result.
  - Actions are built on topics and services. Their functionality is similar to services, except actions can be canceled. They also provide steady feedback, as opposed to services which return a single response.
  - Actions use a client-server model, similar to the publisher-subscriber model (described in the topics tutorial). An “action client” node sends a goal to an “action server” node that acknowledges the goal and returns a stream of feedback and a result.
  - 1:1 통신, 명령을 보내고 중간 처리 과정 및 처리 결과를 수신하는 통신 방식
  - 중간에 명령 취소 가능



참조: Understanding actions

```
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
```

```
$ ros2 action list -t
$ ros2 action info /turtle1/rotate_absolute
$ ros2 interface show turtlesim/action/RotateAbsolute

Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/ros_tutorial/turtlesim_rot_client.py

$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 launch ros_tutorial turtlesim.launch.py
$ ros2 run ros_tutorial turtlesim_rot_client
```

```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'turtlesim_circle = ros_tutorial.turtlesim_circle:main',
            'turtlesim_echo = ros_tutorial.turtlesim_echo:main',
            'turtlesim_abs_client = ros_tutorial.turtlesim_abs_client:main',
            'turtlesim_abs_server = ros_tutorial.turtlesim_abs_server:main',
            'turtlesim_rot_client = ros_tutorial.turtlesim_rot_client:main',
        ],
    },
)
```

setup.py

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from rclpy.action import ActionClient
from turtlesim.action import RotateAbsolute


class TurtlesimRotateClient(Node):
    def __init__(self):
        super().__init__('turtlesim_rot_client')

        self.client = ActionClient(self, RotateAbsolute, '/turtle1/rotate_absolute')
        self.get_logger().info('action client started...')

    def send_goal(self, theta):
        goal_req = RotateAbsolute.Goal()
        goal_req.theta = theta

        if self.client.wait_for_server(10) is False:
            self.get_logger().info('service not available...')
            return

        goal_future = self.client.send_goal_async(goal_req, feedback_callback=self.feedback_callback)
        goal_future.add_done_callback(self.goal_callback)

    def goal_callback(self, future):
        self.goal_handle = future.result()

        if not self.goal_handle.accepted:
            self.get_logger().info('gole rejected...')
            return

        self.get_logger().info('gole accepted...')
        goal_future = self.goal_handle.get_result_async()
        goal_future.add_done_callback(self.result_callback)
        # cancel test
        # self.timer= self.create_timer(1.0, self.timer_callback)

    def feedback_callback(self, msg):
        feedback = msg.feedback
        self.get_logger().info(f'recv feedback: {feedback.remaining}')

    def result_callback(self, future):
        result_handle = future.result()
        res = result_handle.result
        self.get_logger().info(f'recv result: {res.delta}')

        self.destroy_node()
        rclpy.shutdown()

    def timer_callback(self):
        self.get_logger().info(f'canceling goal...')

        cancel_future = self.goal_handle.cancel_goal_async()
        cancel_future.add_done_callback(self.cancel_callback)
        # stop timer
        self.timer.cancel()

    def cancel_callback(self, future):
        result_handle = future.result()
        if len(result_handle.goals_canceling) > 0:
            self.get_logger().info('canceling goal success...')
        else:
            self.get_logger().info('canceling goal fail...')

        self.destroy_node()
        rclpy.shutdown()

def main(args=None):
    rclpy.init(args=args)

    client = TurtlesimRotateClient()
    client.send_goal(3.14)

    rclpy.spin(client)


if __name__ == '__main__':
    main()
```

turtlesim_rot_client.py

Edit src/ros_tutorial/setup.py
Edit src/ros_tutorial/ros_tutorial/turtlesim_rot_server.py

$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 run ros_tutorial turtlesim_rot_server
$ ros2 run ros_tutorial turtlesim_rot_client

```python
import os
from glob import glob
from setuptools import setup

package_name = 'ros_tutorial'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    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/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='cchyun',
    maintainer_email='cchyun@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'turtlesim_circle = ros_tutorial.turtlesim_circle:main',
            'turtlesim_echo = ros_tutorial.turtlesim_echo:main',
            'turtlesim_abs_client = ros_tutorial.turtlesim_abs_client:main',
            'turtlesim_abs_server = ros_tutorial.turtlesim_abs_server:main',
            'turtlesim_rot_client = ros_tutorial.turtlesim_rot_client:main',
            'turtlesim_rot_server = ros_tutorial.turtlesim_rot_server:main',
        ],
    },
)
```

setup.py

```python
#!/usr/bin/env python3

import time
import rclpy
from rclpy.node import Node
from rclpy.action import ActionServer, GoalResponse, CancelResponse
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.executors import MultiThreadedExecutor
from turtlesim.action import RotateAbsolute

class TurtlesimRotateServer(Node):
    def __init__(self):
        super().__init__('turtlesim_rot_server')

        self.server = ActionServer(self, RotateAbsolute,
                                   '/turtle1/rotate_absolute',
                                   callback_group=ReentrantCallbackGroup(),
                                   execute_callback=self.execute_callback,
                                   goal_callback=self.goal_callback,
                                   cancel_callback=self.cancel_callback)
        self.get_logger().info('action server started...')


    def goal_callback(self, goal_request):
        self.get_logger().info(f'recv goal request: {goal_request}')
        return GoalResponse.ACCEPT

    def cancel_callback(self, cancel_request):
        self.get_logger().info(f'recv cancel request: {cancel_request}')
        return CancelResponse.ACCEPT


    async def execute_callback(self, goal_handle):
        feedback = RotateAbsolute.Feedback()
        feedback.remaining = 10.0
        for i in range(10):
            if goal_handle.is_cancel_requested:
                goal_handle.canceled()
                self.get_logger().info('action canceled...')
                return RotateAbsolute.Result()

            feedback.remaining -= 1
            goal_handle.publish_feedback(feedback)
            time.sleep(1)

        goal_handle.succeed()
        self.get_logger().info('action succeed...')

        res = RotateAbsolute.Result()
        res.delta = 0.0
        return res


def main(args=None):
    rclpy.init(args=args)

    server = TurtlesimRotateServer()
    executor = MultiThreadedExecutor()
    rclpy.spin(server, executor=executor)

    server.destroy()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

turtlesim_abs_server.py

Thanks