Robot Operating System 2

건국대학교 스마트운행체공학과 **21**학번 유승민

2025.11.12

What is ROS?

- **Operating System:** 운영체제 **(Windows, IOS)**
  ROS: 로봇 운영체제

- 로봇을 위한 **Software Platform**
- 로봇 개발에 필요한 통신, 제어, 센서 처리, 시각화, 시뮬레이션을 쉽게 구성할 수 있도록 도와줌

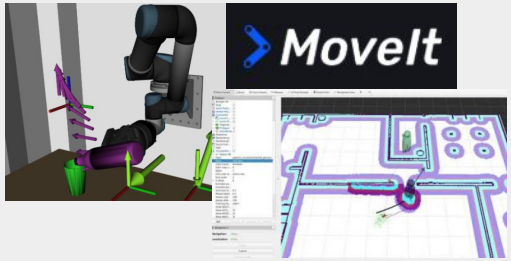
- **Why ROS?**
  수많은 패키지**(OpenCV, Nav2, MoveIt, SLAM** 등**)**를 공유
  기본적인 아키텍쳐를 구축해 놓으면 다른 로봇에도 쉽게 이식
  + 오픈소스가 많다.

★★★★★
ROS는 message system으로
노드 간 통신을 한다.

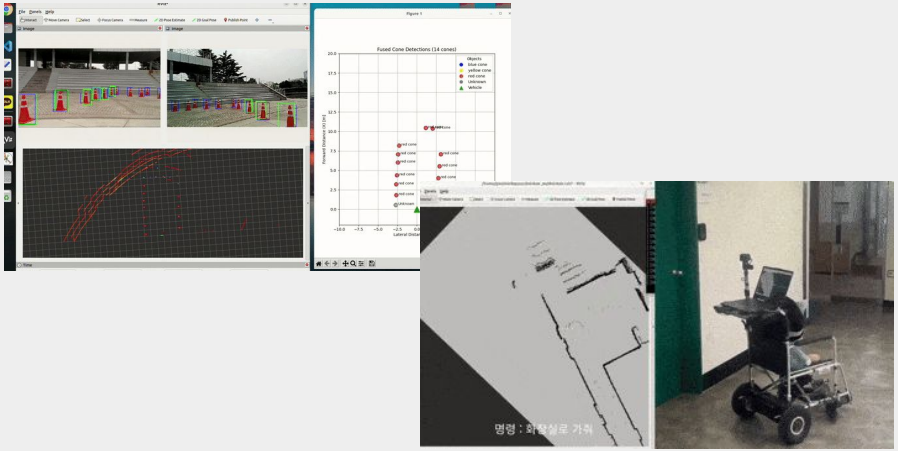
Example

- Hardware
  https://picknik.ai/hardware-ecosystem/

- 많은 패키지 지원



My experience



Terminal & Terminator & vscode

<Linux 터미널 기본 명령어>

yoo@yoo ~$: user id, pc 이름, ~은 경로(home)

ls(list): 디렉토리 파일 목록 list up

cd(change directory): 디렉토리 이동 (cd .. / cd)

mkdir(make directory): 디렉토리 생성

clear: 터미널 지우개

ctrl+shift+c & v : 복사 & 붙여넣기

ctrl+c: 실행 종료

cp(copy): 파일 복사. 두 개의 argument 필요

echo: 문자열이나 변수의 값을 출력하는 명령어

gedit or nano: 파일 편집기

chmod(change mode): 파일을 사용할 수 있도록 사용 권한 부여 ex) chmod +x file.txt

Terminator 설치

```
sudo apt update
sudo apt install terminator
```

VScode 설치

 # 1. 필수 패키지 설치
sudo apt update
sudo apt install wget gpg apt-transport-https -y

 # 2. Microsoft GPG 키 등록
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | sudo gpg --dearmor -o /usr/share/keyrings/packages.microsoft.gpg

 # 3. VS Code 저장소 추가
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/packages.microsoft.gpg] https://packages.microsoft.com/repos/code stable main" | sudo tee /etc/apt/sources.list.d/vscode.list

 # 4. 패키지 목록 갱신
sudo apt update

 # 5. VS Code 설치
sudo apt install code -y

Turtlesim example

- gedit 설치 (텍스트 편집기)

```
sudo apt update
sudo apt install gedit -y
```

- Add sourcing to your shell

```
echo “source /opt/ros/humble/setup.bash” >> ~/.bashrc
```

<Turtlesim 구동해보기>

#거북이 생성

```
ros2 run turtlesim turtlesim_node
```

#키보드 입력으로 거북이 조종

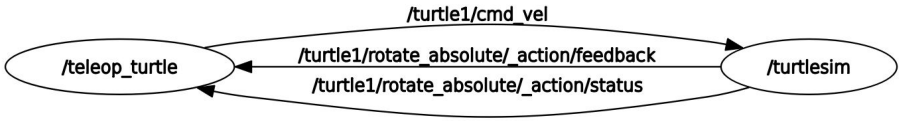
```
ros2 run turtlesim turtle_teleop_key
```

```
rqt_graph
```

<ROS 기본 구조>

1. ROS 특징
   - Peer to Peer: 노드간 통신 (Topic, Service, Action)
   - Multi-Lingual: 노드간 독립적으로 다른 언어로 코드를 짤 수 있다.

2. Node란?
   실행 가능한 최소한의 단위
   하나의 Process라고 생각하면 된다.
   -> Node간 통신으로 로봇 SW를 디자인하게 됨.



Message Communication Between Node

- Message★
  노드간 통신을 할 때 **message** 형태로 주고 받는다.
  기본 **message**는 **ROS2** 패키지 **.msg** 파일에 정의

- Communication method in ROS2
  - Topic & Service & Action

1. ROS Topic ★★★
   단방향, 비동기 통신

<기본 구조>

Publish 노드에서 데이터를 **Topic**으로 발행하고 **Subscribe** 노드는 그 **Topic**을 구독하여 데이터를 받음.

Publisher → Topic → Subsciber

주로 센서 데이터, 상태 정보를 보낼 때 사용

2. ROS Service
   양방향 통신

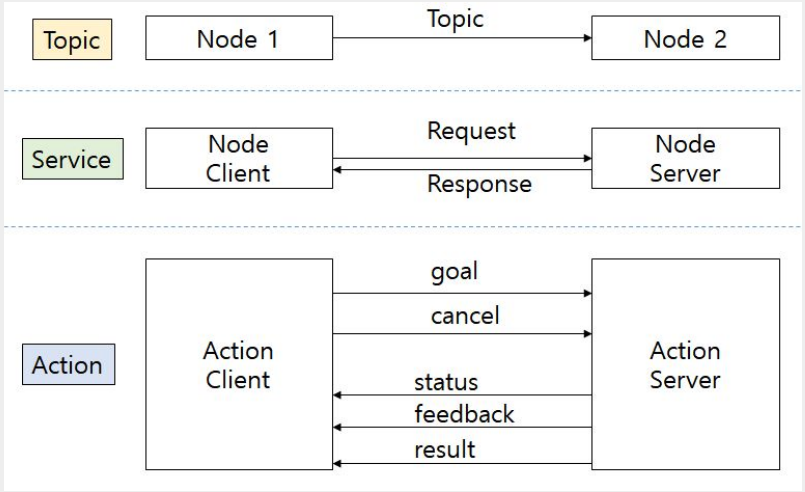
요청(**Request**)과 응답(**Response**) 구조를 가짐

Client → Server: 요청을 보냄

Server → Client: 응답을 보냄

ex) "무언가 해줘" → "알겠어, 완료했어"

→ 요청하고 답을 받는 것 **(ON/OFF,** 설정 등)



Message Communication Between Node

3. ROS Action

   클라이언트와 서버가 미리 정의된 **Action** 프로토콜을 통해 통신하는 방식

   ‘오래 걸리는 작업’을 처리하기 위해 설계된 구조

<기본 구조>

Client → Server: 목표(goal)를 보냄

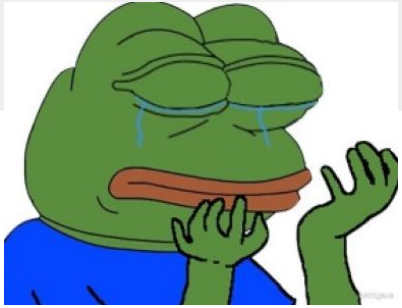
Server → Client: 중간 피드백을 계속 보냄, 최종 결과도 따로 보냄

필요시에 중단(**preempt**) 가능

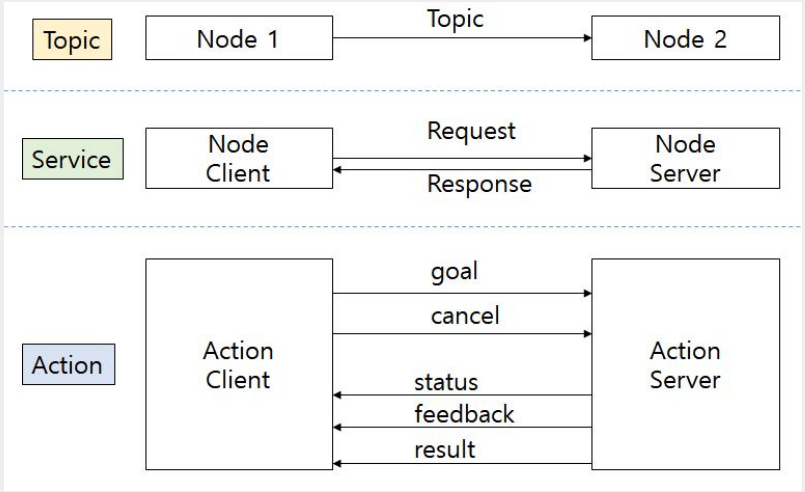
ex) “지금부터 뭘 해줘”, “어떻게 되고 있어?”, “끝났어!”

→ 오래 걸리는 일, 중간 피드백도 받고 싶을 때 (로봇 제어 등)

아.. 너무 어렵다.. **tlqkf**



★ **ROS Topic**만 제대로 이해하고 사용하자. 충분하다.



Turtlesim에서의 Message&Topic

ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
rqt_graph

#실행중인 토픽 list up
ros2 topic list / ros2 topic list -t

#echo
ros2 topic echo <토픽명>

#info
ros2 topic info <토픽명>

#type
ros2 topic type <토픽명>

```
yoo@yoo:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

```
yoo@yoo:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
```

```
yoo@yoo:~$ ros2 topic type /turtle1/cmd_vel
geometry_msgs/msg/Twist
```

#궁금한 메시지 타입(구조) 확인
ros2 interface show <메시지 타입>

cmd_vel: command of velocity

Twist 데이터 타입이란?
- 3차원짜리 벡터 두개 linear, angular (roll, pitch, yaw)

pose: 위치 정보
ros2 topic echo /turtle1/pose

Topic publish once
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear:{x: 2,y: 0,z: 0}, angular:{x: 0,y: 0,z: 1.8}}"

```
yoo@yoo:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its
ar parts.

Vector3  linear
        float64 x
        float64 y
        float64 z
Vector3  angular
        float64 x
        float64 y
        float64 z
```

Create your own ROS2 PKG

- Package란?
  ROS에서는 기능을 하나의 파일로 사용하지 않는다.
  기능별로 폴더(패키지)를 만들어서 관리한다.
  예를 들어 로봇에서

  - 카메라 처리하는 기능 → 하나의 패키지
  - 모터 제어하는 기능 → 하나의 패키지
  - SLAM 하는 기능 → 또 하나의 패키지

- make your own PKG (의존성 추가 ★)

```
mkdir ros2_ws/src
cd ros2_ws
cd src
```

```
yoo@yoo:~/workspace/edu_ws/src/auto_turtle2$ ls
CMakeLists.txt  include  package.xml  src
```

```
ros2 pkg create --build-type ament_python <package_name>
ros2 pkg create --build-type ament_cmake <package_name>
```

- package.xml
  패키지의 데이터를 정의해주는 파일
  ex) 패키지 이름, 유지보수 담당자, 코드 배포 시 사용하는 라이센스, 빌드 도구 의존성, 테스트에 필요한 의존 패키지, 빌드 방식 명시

1. 패키지의 이름 부분은 절대 바뀌면 안된다.
2. 빌드에 필요한 의존성(Dependencies)은 <depend>에 넣는다. 안하면 빌드 후 실행이 안된다.
3. 어떤 패키지를 설치하고 어떤 메시지를 인식하고 어떤 헤더를 포함하는지 ros2 빌드 시스템이 알 수 있게 하기 위해

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
<package format="3">
  <name>auto_turtle1</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="smzzang21@konkuk.ac.kr">yoo</maintainer>
  <license>TODO: License declaration</license>

  <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>
```

e/edu_ws/src/auto_turtle2/package.xml

Create your own ROS2 PKG

- **CmakeLists.txt (for C based language)**
  패키지를 빌드하기 위한 CMake 빌드 시스템의 입력 파일

1. project(my_package): 패키지의 이름

2. find_package(ament_cmake REQUIRED) /
   find_package(rclcpp REQUIRED)
   : 빌드에 필요한 의존 패키지를 명시

3. add_executable(my_node src/my_node.cpp):
   소스 코드를 빌드해서 실행 파일 생성. 사용권한 부여

   add_executable([노드명] [패키지 내 경로])

```
cmake_minimum_required(VERSION 3.8)
project(auto_turtle2)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all sour
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
[EOF]
```

- **setup.py(for python based package)**

  이 패키지를 어떻게 설치하고 실행할지 정의하는 설계서

  setup.py의 entry_points에 노드를 추가해줘야 빌드 후 실행이 가능하다.

```
entry_points={
    'console_scripts': [
        'auto_pub = auto_turtle1.auto_pub:main',
    ],
},
```

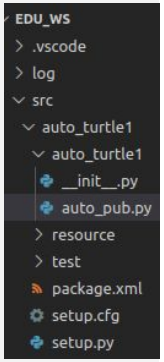
Move Turtle by topic

GOAL: 네모 네모 거북이

publish: loop 이용하여 사각형으로 거북이 움직이기

subscribe: 거북이 **pose** 받아서 중심과의 거리 출력

1. **Publisher** 만들기



Node 기본형태

```python
import rclpy
from rclpy.node import Node

class AutoPublisher(Node):
    def __init__(self):
        super().__init__('square_move') # 노드 이름 설정

def main(args=None):
    rclpy.init(args=args)
    node = AutoPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()[EOF]
```

Publisher 만들기

```python
self.pub = self.create_publisher(Twist, 'turtle1/cmd_vel', 10) # 퍼블리셔
```

msg에 내용 담아서 **publish**

```python
msg = Twist() # Twist 메시지 객체 생성
```

```python
msg.linear.x = 1.0
msg.angular.z = 0.0
self.pub.publish(msg) # 메시지 퍼블리시
```

package.xml & setup.py 설정

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

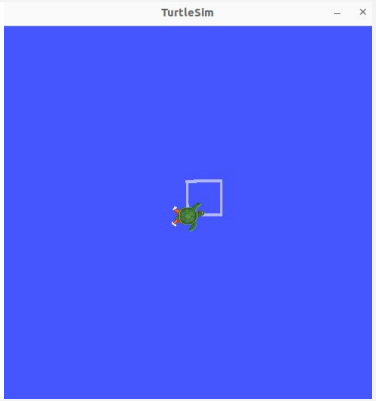
```
entry_points={
    'console_scripts': [
        'auto_pub = auto_turtle1.auto_pub:main',
    ],
},
```

Move Turtle by topic

```
ros2 run turtlesim turtlesim_node
rqt_graph
ros2 topic echo /turtle1/cmd_vel
```

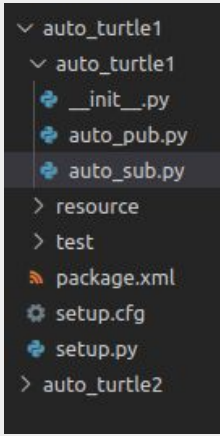
2. 빌드 후 실행

```
colcon build
source install/setup.bash

ros2 run <package명> <node명>
ros2 run auto_turtle1 auto_pub

ros2 topic echo /turtle1/cmd_vel
```



숫자 바꿔가며.. **build**는 꼭 다시 해줘야 함.

2. Subscriber 만들기



Node 기본형태

```python
import rclpy
from rclpy.node import Node

class AutoSubscriber(Node):
    def __init__(self):
        super().__init__('pose_calc_distance')  # 노드 이름 설정


def main(args=None):
    rclpy.init(args=args)
    node = AutoSubscriber()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()[EOF]
```

Goal

위치를 받아서 사분면 중심과의 거리 출력

center = (5.5,5.5)

Calculate distance

Subscriber 만들기

```
self.subscription = self.create_subscription( # 구독자 생성
    Pose,
    'turtle1/pose',
    self.pose_callback,
    10)
```

callback 함수가 존재!

```
def pose_callback(self, msg):
    #현재 좌표
    turtle_x = msg.x
    turtle_y = msg.y

    # 중심과의 거리 계산
    distance = math.sqrt((turtle_x - self.center_x) ** 2 + (turtle_y - self.center_y) ** 2)
    self.get_logger().info(f'거리: {distance:.2f}')
```

여기서 msg.x와 msg.y의 의미는? type은?
= Pose의 .x와 .y field를 쓰겠다.

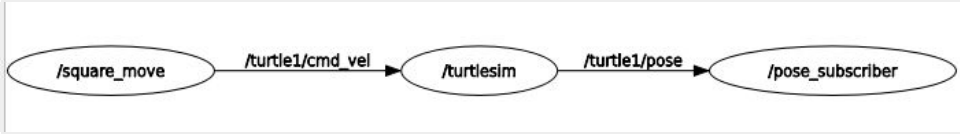
package.xml & setup.py 설정

```
entry_points={
    'console_scripts': [
        'auto_pub = auto_turtle1.auto_pub:main',
        'auto_sub = auto_turtle1.auto_sub:main',
    ],
},
```

#빌드 후 실행

colcon build
source install/setup.bash

ros2 run <package명> <node명>
ros2 run auto_turtle1 auto_sub



Goal & Summary

1. 터미널에 익숙해지자

2. ROS는 message system으로 노드 간 통신을 한다.

3. ROS의 기본적인 패키지 사용 방법을 이해한다.

4. ROS Topic만 제대로 이해하고 사용하자. 충분하다.

5. 기본적인 Publisher & Subscriber을 만들 수 있다.

Homework

1. **pose_calc_distance** 노드에 **publisher** 추가하여 **/distance** 토픽으로 **publish**하기

2. **square_move** 노드에서 무한으로 움직이는 거북이를 **stop topic(type** 상관 없음**)**들어오면 거북이 멈추게 하기.

Thank you