Lecture #5
ROS2의 개발 툴, Node

경희대학교 기계공학과
로봇 제어 지능 연구실
김상현 교수

01 Basic Concept

Basic Concept

- ROS2는 공식적으로 C(C++), Python3를 사용, 그 외 비공식적으로 C# 및 JavaScript, Rust 등도 사용 가능.
  - 본 강의에서는 Python를 위주로 학습 예정

- ROS2는 공식적으로 Cmakelist.txt, Setup.py를 사용한 Compiler을 활용함.
  - 특히, “colcon”이란 Complier를 사용

- 본 강의에서 사용하는 IDE는 VisualStudioCode를 활용할 예정
  - https://code.visualstudio.com/Download 참고

02 ROS2 Node

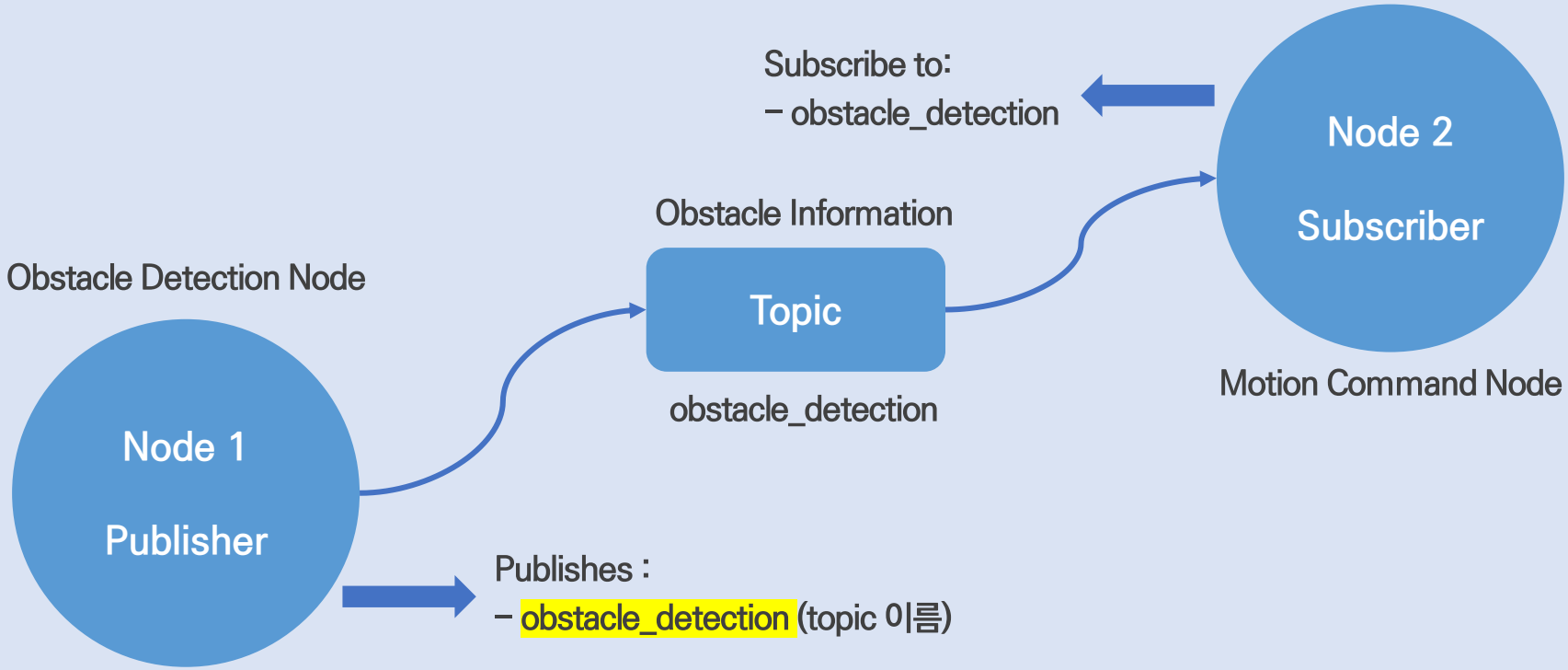
ROS2 Node

- 노드(Node)는 Ros2에서 최소 단위의 실행 가능한 프로세스를 가리키는 용어

- 예시 (ex. 장애물을 감지하고 운동 명령을 생성하는 간단한 로봇)

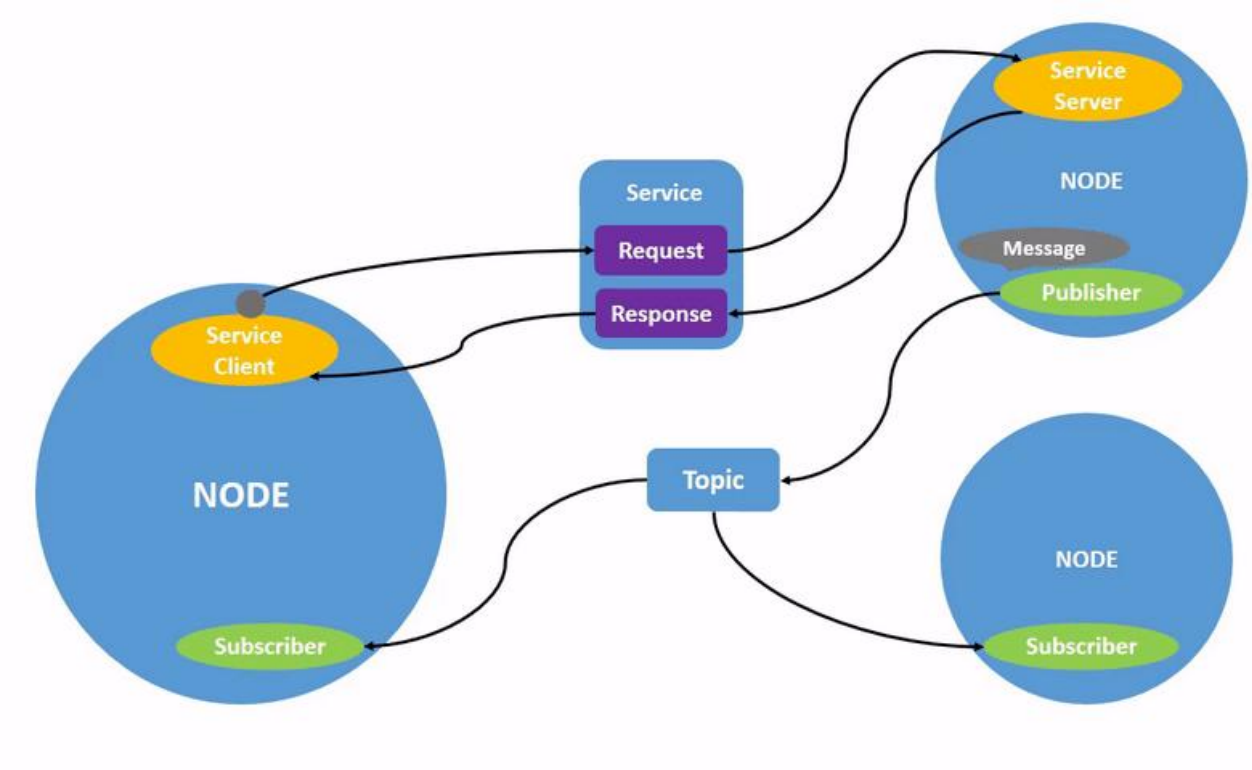
Node 1: 장애물 감지 노드 –카메라나 LiDAR와 같은 센서로부터 데이터를 수집하여 로봇의 환경 내 장애물을 감지하는 역할

Node 2: 모션 명령 노드 – “Obstacle_detection” topic을 subscribe 하여 장애물에 대한 정보를 받아 구동



ROS2 Node

- 예시 (ex. 다중 노드 로봇 응용 프로그램)



- 노드와 노드 사이에 입출력 데이터를 주고 받고 이를 Message라고 한다.
- Message를 주고받는 통신 방법에 따라 토픽(Topic), 서비스(Service), 액션(Action)으로 나눌 수 있다.

ROS2 Node

ROS2 노드 실행법

기본적인 실행법

$ros2 run ⟨package_name⟩ ⟨executable_name⟩

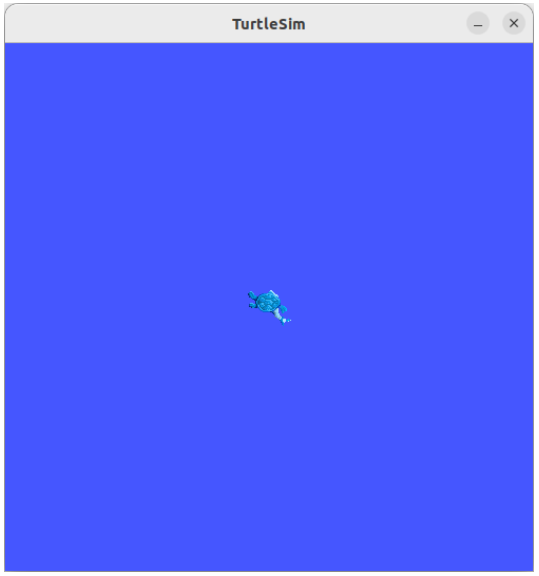
패키지 이름 실행가능한 파일

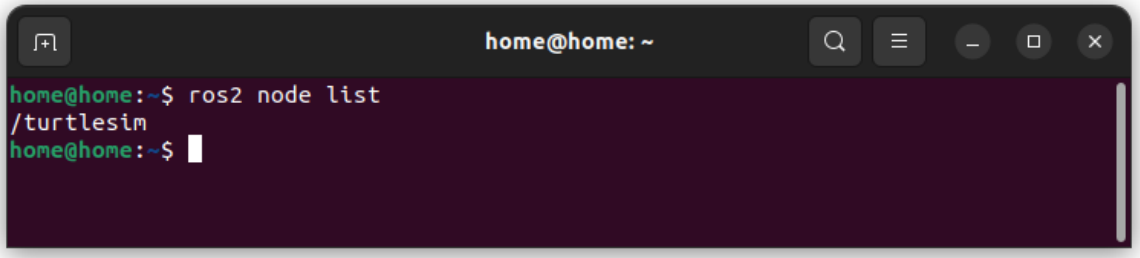
Turtlesim Node 실행 (Terminal 1)

$ ros2 run turtlesim turtlesim_node

실행중인 Node 목록 확인 (Terminal 2)

$ ros2 node list





ROS2 Node

ROS2 노드 실행법

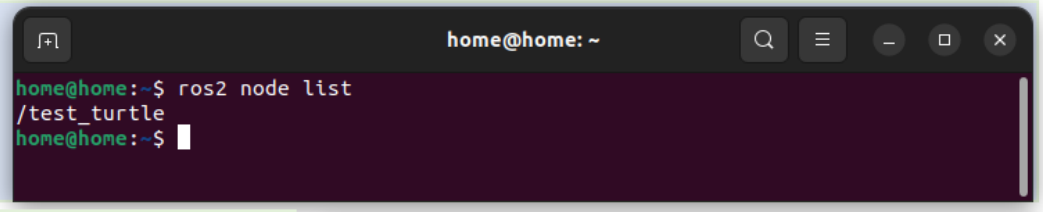
Turtlesim Node 이름 바꿔서 실행 (Remapping)

기존 Terminal 1에서 Ctrl + C를 눌러 노드를 종료한 후
$ ros2 run turtlesim turtlesim_node --ros-args --remap __node:=test_turtle

Remapping을 사용하면 사용자가 노드 이름, 토픽 이름, 서비스 이름들을 사용자 정의 값으로 재할당이 가능함

실행중인 Node 목록 확인 (Terminal 2)

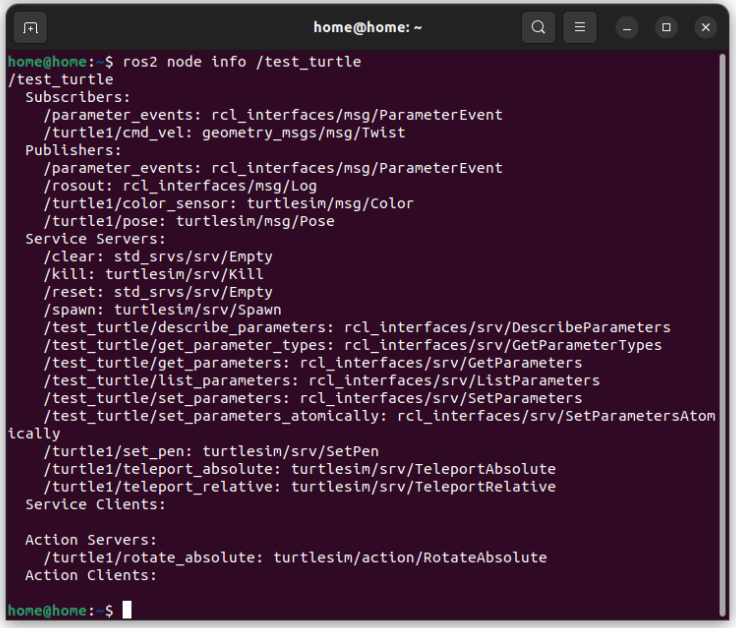
$ ros2 node list



실행중인 Node 정보 확인(Terminal 2)

$ ros2 node info /test_turtle

info 명령어는 해당 노드와 관련된 모든
subscribers, publishers, services, actions
목록을 발행한다



03 ROS2 build system

ROS2 Build System

- 각 소스코드들은 package에서 관리하여, 하나의 패키지들은 1개 이상의 소스 코드를 빌드하게 됨.
  - colcon은 ROS2의 공식 build system

- ROS의 파일 시스템

Package는 로스의 메인 Unit!

1) ROS의 runtime process들을 관리 (node)
2) 해당 node들의 dependency를 관리
3) Dataset 등을 관리
4) 알고리즘의 Configuration files 들을 관리

- 패키지의 관리
  - 커스텀 패키지들은 사용자의 workspace에서 만듦 (주로 ros2_ws로 생성)

```
workspace_folder/
    src/
        package_1/
            CMakeLists.txt
            package.xml
        ...
        package_n/
            CMakeLists.txt
            package.xml
```

다음처럼 각 package들은 CmakeLists.txt와 package.xml을 포함한다.

ROS2 Build System

실습 #1: ros2_ws 생성 및 등록

Workspace 생성

```
$ mkdir ros2_ws  # catkin_ws의 생성
$ cd ros2_ws # catkin_ws로의 이동
$ mkdir src # src 폴더의 생성
$ cd .. # 다시 ros2_ws로 이동
$ colcon build
```

Workspace 등록

Catkin_ws를 해당 터미널에 등록

```
$ source install/setup.bash
```

Workspace 자동등록

ros2_ws를 영구적으로 등록 (혹은 gedit ~/.bashrc에서 직접 source $HOME/ros2_ws/install/setup.bash 추가 해도 됨)

```
$ echo " source /$HOME/ros2_ws/install/setup.bash " >> ~/.bashrc
$ source ~/.bashrc
```

ROS2 Build System (Python)

실습 #2: ros2 build를 활용한 hello_ros_py 실습

```
$ cd ros2_ws/src
$ ros2 pkg create hello_ros_py  --build-type ament_python --dependencies rclpy
```

hello_ros_py의 setup.py 파일을 수정

```
entry_points={
        'console_scripts': [
            'hello_world_node = hello_ros_py.practice2:main',
          ],
```

Node를 등록하는 과정
hello_ros_py패키지의 practice2.py파일의
main함수가 실행되도록 설정

ROS2 Build System (Python)

- 실습 #2: ros2 build를 활용한 hello_ros_py 실습

setup.py 파일의 구성

```
from setuptools import find_packages, setup

package_name = 'hello_ros_py'
setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Sanghyun Kim',
    maintainer_email='kim87@khu.ac.kr',
    description=' Ros Build System Tutorial ',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
          'hello_world_node = hello_ros_py.practice2:main',
        ], },
)
```

필요한 함수 Import

패키지 이름 정의

Setup함수의 구성

name = 패키지 이름 구성
version = 패키지 버전 지정
packages = 빌드에 필요한 package 불러오기
data_file = 패키지와 관련된 데이터 파일 설치 위치
install_requires = 패키지 설치에 필요한 의존성 명시
zip_safe = 압축된 상태로의 설치 가능 명시
maintainer ~ email = 패키지 관리자 명시
description = 패키지에 대한 설명
license = 패키지 라이선스
tests_require = 테스트에 필요한 패키지 명시
entry_points = 실행 파일 설정

ROS2 Build System (Python)

- 실습 #2: ros2 build를 활용한 hello_ros_py 실습

package.xml 파일의 구성

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/
schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>hello_ros_py</name>
  <version>0.0.0</version>
  <description> Ros Build System Tutorial </description>
  <maintainer email="kim87@khu.ac.kr">Sanghyun Kim</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</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>
```

package의 기본 설정 package 이름 ..

의존성 설정

테스트 의존성

build type같은 패키지 설정

ROS2 Build System (Python)

- 실습 #2: ros2 build를 활용한 hello_ros_py 실습

practice2.py를 hello_ros_py/hello_ros_py 폴더에 생성

```
import rclpy
from rclpy.node import Node

class HelloWorldNode(Node):
    def __init__(self):
        super().__init__('hello_world_node')


def main(args=None):
    rclpy.init(args=args)
    node = HelloWorldNode()
    node.get_logger().info('Hello World!')
    node.destroy_node()
    rclpy.shutdown()

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

모듈 Import

hello_world_node 생성

main 함수 지정

스크립트가 실행될 때 main함수 호출되도록 설정

ROS2 Build System (C++)

실습 #2: ros2 build를 활용한 hello world 실습

```
$ cd ros2_ws/src
$ ros2 pkg create hello_ros_cpp --build-type ament-cmake --dependencies rclcpp
```

CMakelists.txt를 구성(중요부분)

```
cmake_minimum_required(VERSION 3.8)

project(hello_ros_cpp)

find_package(ament_cmake REQUIRED)

find_package(rclcpp REQUIRED)

add_executable(hello_world_node src/practice2.cpp)

ament_target_dependencies(hello_world_node rclcpp)

install(TARGETS  hello_world_node  DESTINATION
lib/hello_ros_cpp)

ament_package()
```

Package의 기본 버전 및 이름 설정
package.xml의 〈name〉과 일치!

ament 빌드할 때 필요 dependencies 지정

실행 파일 지정 및 dependencies

빌드 대상을 설치 (Node, Launch …)

ROS2 Build System (C++)

- 실습 #2: ros2 build를 활용한 hello world 실습

```
$ cd ros2_ws/src
$ mkdir hello_ros_cpp
$ cd hello_ros_cpp
```

package.xml를 hello_ros 폴더에 생성

```
<?xml version="1.0"?>
<package format="3">
  <name>hello_ros_cpp</name>
  <version>0.0.0</version>

  <description>Ros Build System Tutorial</description>
  <maintainer  email="kim87@khu.ac.kr">Sanghyun Kim</maintainer>
  <license>TODO</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>

</package>
```

Xml 버전 설정 및 package 이름 및 버전 설정

Package 설명

Build 방식 선택 및 dependencies지정

ROS2 Build System (C++)

- 실습 #2: ros build를 활용한 hello world 실습

pratice2.cpp를 hello_ros_cpp/src 폴더에 생성

```
#include "rclcpp/rclcpp.hpp"

class HelloWorldNode : public rclcpp::Node
{
   public:
      HelloWorldNode() : Node("hello_world_node")
      {
      }
};

int main(int argc, char *argv[])
{
   rclcpp::init(argc,argv);
   auto node = std::make_shared<HelloWorldNode>();
   RCLCPP_INFO(node -> get_logger(),"Hello World!");
   rclcpp::shutdown();
}
```

Ros2 C++ Library 추가

Class를 통해 Node 구현

Main문 생성

ROS2 Build System

- 실습 #2: ros2 build를 활용한 hello_world_py & hello_world_cpp 실습

```
$ cd ~/ros2_ws
$ colcon build
```

hello_ros_py의 hello_world_node 실행

```
$ros2 run hello_ros_py hello_world_node
```

hello_ros_cpp의 hello_world_node 실행

```
$ros2 run hello_ros_cpp hello_world_node
```

04 ROS2 Logging System

ROS2 Logging System (Python)

- ROS는 기본적으로 DEBUG, INFO, WARN, ERROR, FATAL로 분류하여 LOG를 쓸 수 있음.

ROS Debug의 예시

```
self.get_logger().info("INFO Log");
self.get_logger().debug("DEBUG Log");
self.get_logger().warn("WARN Log");
```

- 실습 #3: 앞 서 만든 패키지에 practice3.py를 만든 뒤 Debug, Condition 등을 활용해 본다.

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

class LoggingNode(Node):
    def __init__(self):
        super().__init__('logging_node')
        self.i = 1
        self.timer = self.create_timer(1.0, self.timer_callback)  # 1초 주기로 실행

    def timer_callback(self):
        self.get_logger().info(f'Counted to {self.i}')
        if self.i % 3 == 0:
            self.get_logger().info('is divisible by 3.')
        elif self.i % 5 == 0:
            self.get_logger().debug('is divisible by 5.')
        elif self.i % 7 == 0:
            self.get_logger().warn('is divisible by 7.')
        elif self.i % 11 == 0:
            self.get_logger().error('is divisible by 11.')
        elif self.i % 13 == 0:
            self.get_logger().fatal('is divisible by 13.')

        self.i += 1

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

    node = LoggingNode()
    rclpy.spin(node)

    node.destroy_node()
    rclpy.shutdown()

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

Timer을 통해 1초마다 timer_callback이 실행되도록 설정

setup.py 추가사항

```
entry_points={
        'console_scripts': [
            'hello_world_node = hello_ros_py.practice2:main',
            'logging_node = hello_ros_py.practice3:main'
        ],
```

Node 추가

ROS2 Logging System (C++)

- ROS는 기본적으로 DEBUG, INFO, WARN, ERROR, FATAL로 분류하여 LOG를 쓸 수 있음.

ROS Debug의 예시

```
RCLCPP_INFO(node->get_logger(), "Hello %s", "World");
RCLCPP_DEBUG (node->get_logger(), "Hello " << "World");
RCLCPP_WARN (node->get_logger(), “this is bad");
```

- 실습 #3: 앞 서 만든 패키지에 practice3.cpp를 만든 뒤 Debug, Condition 등을 활용해 본다.

```cpp
#include "rclcpp/rclcpp.hpp"

class LoggingNode : public rclcpp::Node
{
public:
    LoggingNode() : Node("logging_node"), i_(1)
    {
        timer_ = this->create_wall_timer(
            std::chrono::seconds(1),
            std::bind(&LoggingNode::timer_callback, this));
    }
private:
    void timer_callback()
    {
        RCLCPP_INFO(this->get_logger(), "Counted to %d", i_);
        if (i_ % 3 == 0)
            RCLCPP_INFO(this->get_logger(), "is divisible by 3.");
        else if (i_ % 5 == 0)
            RCLCPP_DEBUG(this->get_logger(), "is divisible by 5.");
        else if (i_ % 7 == 0)
            RCLCPP_WARN(this->get_logger(), "is divisible by 7.");
        else if (i_ % 11 == 0)
            RCLCPP_ERROR(this->get_logger(), "is divisible by 11.");
        else if (i_ % 13 == 0)
            RCLCPP_FATAL(this->get_logger(), "is divisible by 13.");
        i_++;
    }
    rclcpp::TimerBase::SharedPtr timer_;
    int i_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<LoggingNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

Timer을 통해 1초마다 timer_callback이 실행되도록 설정

CMakeLists.txt 추가사항

```
add_executable(logging_node src/practice3.cpp)
ament_target_dependencies(logging_node rclcpp)

install(TARGETS
  hello_ros_node
  logging_node
  DESTINATION lib/hello_ros_cpp)
```

Node 추가

ROS2 Logging System

- 실습 #2: 앞서 만든 logging_node 실행

```
$ cd ~/ros2_ws
$ colcon build
```

hello_ros_py의 logging_node 실행

```
$ros2 run hello_ros_py logging_node
```

hello_ros_cpp의 hello_world_node 실행

```
$ros2 run hello_ros_cpp logging_node
```

감사합니다