ROS 2 스터디 7주차

구본우

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.1 코드 스타일 가이드

- 오픈소스 소프트웨어는 커뮤니티의 협업을 기반으로 한 공동의 결과물이다.

- ROS도 ROS 커뮤니티의 공동의 결과물로 협업이 기반이다.

- ROS2 developer guide, ROS Enhancement Proposals(REPs)와 같은 가이드와 규칙도 만들고, ROS 2 Code style과 같이 일관된 코드 스타일을 지키기 위하여 사용되는 각 언어에 대해 스타일 가이드라인을 세워 구성원들의 합의하에 이를 따르고 있다.

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.2 기본 이름 규칙

- snake_case: 파일 이름 및 변수명, 함수명 규칙 (소문자)
- CamelCased: 타입 및 클래스 규칙
- ALL_CAPITALS: 상수 규칙
- 단, ROS 인터페이스 파일은 /msg, /srv, /action 폴더에 위치시키며 인터페이스 파일명은 CamelCased 규칙을 따른다. (h(pp) 및 모듈로 변환한 후 구조체 및 타입으로 사용되기 때문)

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.2 기본 이름 규칙 (예외)

- Package.xml
- CmakeLists.txt
- README.md
- LICENSE
- CHANGELOG.rst
- .gitignore
- .travis.yml
- *.repos

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.3 C++ Style

- C++ 코드 스타일은 오픈소스 커뮤니티에서 가장 널리 사용 중인 Google C++ Style Guide를 사용하고 있으며 ROS의 특성에 따라 일부를 수정해 사용하고 있다.

1. 기본 규칙
   - C++14 Standard를 준수한다.
2. 라인 길이
   - 최대 100문자
3. 이름 규칙 (Naming)
   - CamleCased, smake_case, ALL_CAPITALS만을 사용한다.
   - 소스 파일은 cpp 확장자를 사용한다.
   - 헤더 파일은 hpp 확장자를 사용한다.
   - 전역변수(Global variable)를 반드시 사용해야 할 경우 접두어(g_)를 붙인다.
   - 클래스 멤버 변수(Class member variable)는 마지막에 밑줄(_)을 붙인다.

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.3 C++ Style

4. 공백 문자 대 탭 (Spaces vs Tabs)
   - 기본 들여쓰기(Indent)는 공백 문자(Space) 2개를 사용한다. (Tab문자 사용 금지)
   - Class의 접근 지정자(public, protected, private)는 들여쓰기를 하지 않는다.
5. 괄호 (Brace)
   - If, else, do, while, for 구문에 괄호를 사용한다.
   - 괄호 및 공백 사용은 다음 예제를 참고하자. (p.251~252)
6. 주석 (Comments)
   - 문서 주석은 /** */을 사용한다.
   - 구현 주석은 //을 사용한다.
7. 린터 (Linters) (오류검출)
8. 기타

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.4 Python Style

- 파이썬 코드 스타일은 Pthton Engancement Proposals (PEPs)의 PEP 8을 준수한다.

1. 기본 규칙
   - 파이썬 3 (파이썬 3.5 이상)을 사용한다.

2. 라인 길이
   - 최대 100문자

3. 이름 규칙 (Naming)
   - CamleCased, smake_case, ALL_CAPITALS만을 사용한다.

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.4 Python Style

4. 공백 문자 대 탭 (Spaces vs Tabs)
   - 기본 들여쓰기(Indent)는 공백 문자(Space) 4개를 사용한다. (Tab문자 사용 금지)
   - Hanging indent(문장 중간에 들여쓰기를 사용하는 형식)의 사용 방법은 다음 예제 참고 (p.254)
   - 괄호 및 공백 사용은 다음 예제를 참고 (p.254)
5. 괄호 (Brace)
   - 계산식 및 배열 인덱스로 사용하며, 자료형에 따라 적절한 괄호를 사용한다.
   - List = [1, 2, 3, 4, 5] dictionary = {'age': 30, 'name': '홍길동'} tuple = (1, 2, 3, 4, 5)
6. 주석 (Comments)
   - 문서 주석은 """을 사용하며 Docstring Conventions을 기술한 PEP 257을 준수한다.
   - 구현 주석은 #을 사용한다.
7. 린터 (Linters)
8. 기타
   - 모든 문자는 큰 따옴표가 아닌 작은 따옴표를 사용하여 표현한다.

1장 ROS 프로그래밍 규칙 (코드 스타일)

1.5 다른 언어

- C 언어는 C99 Standard를 준수하며 PEP-7을 참고하자.
- 자바스크립트 언어는 Airbnb Javascript Style Guide를 참고하자.

2장 ROS 프로그래밍 기초 (파이썬)

2.1 ROS의 Hello World, rclpy 버전

- 프로그래밍 언어의 시작은 화면에 “Hello World” 문구를 출력하는 것이다.
- ROS 에서는 메시지 전송에 더 초점을 둔다.
- 이번 장에서는 파이썬 언어로 ROS 2의 가장 간단한 구조의 토픽, 퍼블리셔, 서브스크라이버를 작성하고 동작시켜보자.

2장 ROS 프로그래밍 기초 (파이썬)

2.2 패키지 생성

- $ ros2 pkg create [패키지 이름] --build-type [빌드 타입] –dependencies [의존하는 패키지1][의존하는 패키지n]
- 패키지 이름과 의존하는 패키지에 rclpy와 std_msgs를 옵션으로 사용하면 ROS 파이썬 클라이언트 라이브러리와 ROS의 표준 메시지 패키지를 사용하겠다는 의미이다.

2.3 패키지 설정

- 2.3.1 패키지 설정 파일(package.xml)
  - 파이썬에서는 ament_python으로 설정하면 된다.
- 2.3.2 파이썬 패키지 설정 파일(setup.py)
  - 해당 설정을 통해 ros2 run 또는 ros2 launch 명령어로 해당 스크립트를 실행시킬 수 있다.
- 2.3.3 파이썬 패키지 환경설정 파일(setup.cfg)
  - 패키지 이름을 기재해야 하고, colcon을 이용하여 빌드하게 되면 지정 폴더에 실행 파일이 생성된다.

2장 ROS 프로그래밍 기초 (파이썬)

2.4 퍼블리셔 노드 작성

첫 구절은 import 구문이다.

Rclpy의 Node 클래스를 사용하며,

퍼블리셔의 QoS 설정을 위하여 QoSProfile 클래스를 사용

메시지 타입은 std_msgs.msg 모듈의 String 메시지 인터페이스를 사용

이 노드의 메인 클래스는 HelloworldPublisher이고 Node 클래스를 상속해 사용할 예정

__init__ 함수 정의

Publish_helloworld_msg 함수 정의

Main 함수 정의

```python
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String


class HelloworldPublisher(Node):

    def __init__(self):
        super().__init__('helloworld_publisher')
        qos_profile = QoSProfile(depth=10)
        self.helloworld_publisher = self.create_publisher(String, 'helloworld', qos_profile)
        self.timer = self.create_timer(1, self.publish_helloworld_msg)
        self.count = 0

    def publish_helloworld_msg(self):
        msg = String()
        msg.data = 'Hello World: {0}'.format(self.count)
        self.helloworld_publisher.publish(msg)
        self.get_logger().info('Published message: {0}'.format(msg.data))
        self.count += 1


def main(args=None):
    rclpy.init(args=args)
    node = HelloworldPublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

2장 ROS 프로그래밍 기초 (파이썬)

2.5 서브스크라이버 노드 작성

첫 구절은 import 구문이다.

퍼블리셔 노드와 완전히 동일

이 노드의 메인 클래스는 HelloworldSubscriber이고 Node 클래스를 상속해 사용할 예정

__init__ 함수 정의

Subscribe_topic_message 함수 정의

Main 함수 정의

```python
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String


class HelloworldSubscriber(Node):

    def __init__(self):
        super().__init__('Helloworld_subscriber')
        qos_profile = QoSProfile(depth=10)
        self.helloworld_subscriber = self.create_subscription(
            String,
            'helloworld',
            self.subscribe_topic_message,
            qos_profile)

    def subscribe_topic_message(self, msg):
        self.get_logger().info('Received message: {0}'.format(msg.data))


def main(args=None):
    rclpy.init(args=args)
    node = HelloworldSubscriber()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

2장 ROS 프로그래밍 기초 (파이썬)

2.6 빌드

- 워크스페이스 내의 모든 패키지 빌드 시 사용되는 명령어는 다음과 같다.

```
$ cd ~/robot_ws && colcon build --symlink-install
```

- 특정 패키지만 빌드할 때 사용되는 명령어는 다음과 같다.

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-select [패키지 이름1] [패키지 이름n]
```

- 특정 패키지의 의존성 패키지들까지도 함께 빌드할때 사용하는 명령어는 다음과 같다.

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-up-to [패키지 이름]
```

2.7 실행

- 각 노드의 실행은 ros2 run 명령어를 사용해 실행하면 된다.

3장 ROS 프로그래밍 기초 (C++)

3.1 ROS의 Hello World, rclcpp 버전

- 프로그래밍 언어의 시작은 화면에 “Hello World” 문구를 출력하는 것이다.
- ROS 에서는 메시지 전송에 더 초점을 둔다.
- 이번 장에서는 파이썬 언어로 ROS 2의 가장 간단한 구조의 토픽, 퍼블리셔, 서브스크라이버를 작성하고 동작시켜보자.

3장 ROS 프로그래밍 기초 (C++)

3.2 패키지 생성

- $ ros2 pkg create [패키지 이름] --build-type [빌드 타입] –dependencies [의존하는 패키지1][의존하는 패키지n]
- 패키지 이름과 의존하는 패키지에 rclcpp와 std_msgs를 옵션으로 사용하면 ROS C++ 클라이언트 라이브러리와 ROS의 표준 메시지 패키지를 사용하겠다는 의미이다.

3.3 패키지 설정

- 3.3.1 패키지 설정 파일(package.xml)
  - C++에서는 ament_cmake으로 설정하면 된다.
- 3.3.2 빌드 설정 파일(CmakeLists.txt)
  - 의존성 패키지의 설정과 빌드 및 설치 관련 설정을 주의해야 한다.

3장 ROS 프로그래밍 기초 (C++)

3.4 퍼블리셔 노드 작성

첫 구절은 include 및 namespace 구문이다.

코드에서 사용되는 std 계열의 헤더를 우선 선언

Rclcpp의 Node 클래스를 사용하기 위한 rclcpp.hpp 헤더 파일과

메시지의 타입인 String 메시지 인터페이스를 사용하기 위한 string.hpp 헤더 파일 포함

Chrono_literals는 추후에 "500ms", "1s"와 같이 시간을 가식성이 높은 문자로 표현하기 위해

Namespace를 사용할 수 있도록 선언

이 노드의 메인 클래스는 HelloworldPublisher이고 Node 클래스를 상속해 사용할 예정

Public 함수 정의

Private 함수 정의

Main 함수 정의

```cpp
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;


class HelloworldPublisher : public rclcpp::Node
{
public:
  HelloworldPublisher()
  : Node("helloworld_publisher"), count_(0)
  {
    auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
    helloworld_publisher_ = this->create_publisher<std_msgs::msg::String>(
      "helloworld", qos_profile);
    timer_ = this->create_wall_timer(
      1s, std::bind(&HelloworldPublisher::publish_helloworld_msg, this));
  }

private:
  void publish_helloworld_msg()
  {
    auto msg = std_msgs::msg::String();
    msg.data = "Hello World: " + std::to_string(count_++);
    RCLCPP_INFO(this->get_logger(), "Published message: '%s'", msg.data.c_str());
    helloworld_publisher_->publish(msg);
  }
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr helloworld_publisher_;
  size_t count_;
};


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

3장 ROS 프로그래밍 기초 (C++)

3.5 서브스크라이버 노드 작성

첫 구절은 include 구문이다.

코드에서 사용되는 std 계열의 헤더를 우선 선언

Rclcpp의 Node 클래스를 사용하기 위한 rclcpp.hpp 헤더 파일과

메시지의 타입인 String 메시지 인터페이스를 사용하기 위한 string.hpp 헤더 파일 포함

Chrono_literals는 추후에 "500ms", "1s"와 같이 시간을 가식성이 높은 문자로 표현하기 위해

Namespace를 사용할 수 있도록 선언

이 노드의 메인 클래스는 HelloworldSubscriber이고 Node 클래스를 상속해 사용할 예정

Public 함수 정의

Private 함수 정의

Main 함수 정의

```cpp
#include <functional>
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using std::placeholders::_1;


class HelloworldSubscriber : public rclcpp::Node
{
public:
  HelloworldSubscriber()
  : Node("Helloworld_subscriber")
  {
    auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
    helloworld_subscriber_ = this->create_subscription<std_msgs::msg::String>(
      "helloworld",
      qos_profile,
      std::bind(&HelloworldSubscriber::subscribe_topic_message, this, _1));
  }

private:
  void subscribe_topic_message(const std_msgs::msg::String::SharedPtr msg) const
  {
    RCLCPP_INFO(this->get_logger(), "Received message: '%s'", msg->data.c_str());
  }
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr helloworld_subscriber_;
};


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

3장 ROS 프로그래밍 기초 (C++)

3.6 빌드

- 워크스페이스 내의 모든 패키지 빌드 시 사용되는 명령어는 다음과 같다.

```
$ cd ~/robot_ws && colcon build --symlink-install
```

- 특정 패키지만 빌드할 때 사용되는 명령어는 다음과 같다.

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-select [패키지 이름1] [패키지 이름n]
```

- 특정 패키지의 의존성 패키지들까지도 함께 빌드할때 사용하는 명령어는 다음과 같다.

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-up-to [패키지 이름]
```

3.7 실행

- 각 노드의 실행은 ros2 run 명령어를 사용해 실행하면 된다.

4장 ROS 2 Tips

4.1 설정 스크립트 (setup script)

- 새로운 패키지를 빌드하였다면 다음 설정 스크립트를 각 터미널 창에서 실행
- $ source ~/robot_ws/install/local_setup.bash

4.2 setup.bash vs local_setup.bash

- 4.2.1 underlay와 overlay
  - Underlay: ROS 2 설치의 기본 패키지와 라이브러리를 포함하는 디렉토리
  - Overlay: 개발자가 추가로 설치한 패키지와 라이브러리를 포함하는 디렉토리

- 4.2.2 Setup.bash 및 local_setup.bash 사용 방법
  - 설정 스크립트라고 부르고 모든 워크스페이스에 존재하며 각 설정 스크립트마다 사용 목적은 조금씩 다르다.
  - Setup.bash: 현재 작업 공간이 빌드될 때 환경에 제공된 다른 모든 작업 공간에 대한 local_setup.bash를 포함하고 있다.
  - Local_setup.bash: 이 스크립트가 위치해 있는 접두사 경로의 모든 패키지에 대한 환경을 설정한다.

4장 ROS 2 Tips

4.3 colcon_cd

- 현재 작업 디렉터리를 패키지 디렉터리로 빠르게 변경 가능
- $ colcon_cd 패키지이름

4.4 ROS_DOMAIN_ID vs Namespace

- ROS 2를 사용하면서 동일 네트워크를 다른 사람들과 공유하고 있다면 다른 연구원들이 사용하고 있는 노드 정보에 쉽게 접근이 가능하며 관련된 데이터를 공유할 수 있게 된다.
- 이런 기능은 멀티 로봇 제어 및 협업 작업 시 매우 편리한데 독립적인 작업을 해야 할 때에는 이 기능이 역으로 불편할 수 있다. 이를 방지하기 위해 3가지 방법은 다음과 같다.
    1. 물리적으로 다른 네트워크를 사용
    2. ROS_DOMAIN_ID를 이용하여 DDS의 domain을 변경
    3. 각 노드 및 토픽/서비스/액션의 이름에 Namespace 추가

4장 ROS 2 Tips

4.4.1 물리적으로 다른 네트워크

- 원천적인 해결 방법은 독립적인 스위치 허브 및 라우터를 사용하여 네트워크를 완전히 분리해주는 것

4.4.2 ROS_DOMAIN_ID

- 동일 네트워크를 다른 사람들과 함께 사용하고 있다면 서로 다른 ROS_DOMAIN_ID를 각 멤버들이 지정하여 쓰면 된다.

4.4.3 ROS 2 Namespace

- ROS 2의 노드는 고유 이름을 가지니 이러한 고유 이름에 Namespace를 붙여 주면 독립적으로 자신만의 네트워크를 그룹화 할 수 있다.

4.5 colcon과 vcstool 명령어 자동 완성 기능 추가

- 이 두 개의 툴의 자동완성 기능을 사용하려면 colcon-argcomplete.bash 파일과 vcs.bash 파일을 소싱해야 한다.

5장 토픽, 서비스, 액션 인터페이스

5.1 ROS 2 인터페이스 신규 작성

- 사용자가 필요로 하는 형태가 아니라면 새로 만들어 사용해야 한다.
- 이 장에서 msg_srv_action_interface_example 패키지를 만들 것이고 이 인터페이스 전용 패키지에는 msg 인터페이스, srv 인터페이스, action 인터페이스를 포함시킬 것이다.

5.2 인터페이스 패키지 만들기

```
$ cd ~/robot_ws/src
$ ros2 pkg create –build-type ament_cmake msg_srv_action_interface_example
$ cd msg_srv_action_interface_example
$ mkdir msg srv action
```

5장 토픽, 서비스, 액션 인터페이스

5.2 인터페이스 파일 생성

5.2.1 ArithmeticArgument.msg 생성

5.2.2 ArithmeticOperator.srv 생성

5.2.3 ArithmeticChecker.acrion 생성

5.2.4 msg, srv, action 인터페이스 비교

5장 토픽, 서비스, 액션 인터페이스

5.3 패키지 설정 파일 (package.xml)

- 일반적인 패키지와 다른 점은 다음과 같이 빌드 시에 DDS에서 사용되는 IDL 생성과 관련한 rosidl_default_generators가 사용된다는 점과 실행 시에 builtin_interface와 rosidl_default_runtime이 사용된다는 점이다.

```
<?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>msg_srv_action_interface_example</name>
  <version>0.6.0</version>
  <description>
    ROS 2 example for message, service and action interface
  </description>
  <maintainer email="passionvirus@gmail.com">Pyo</maintainer>
  <license>Apache 2.0</license>
  <author email="passionvirus@gmail.com">Pyo</author>
  <author email="routiful@gmail.com">Darby Lim</author>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>builtin_interfaces</exec_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

5장 토픽, 서비스, 액션 인터페이스

5.4 빌드 설정 파일 (CMakeLists.txt)

- 일반적인 패키지와 다르게 다음과 같이 Cmake의 set 명령어로 msg, srv, action 파일을 지정하고 rosidl_generate_interfaces에 해당 set들을 기입하면 된다.

```
set(msg_files
  "msg/ArithmeticArgument.msg"
)

set(srv_files
  "srv/ArithmeticOperator.srv"
)

set(action_files
  "action/ArithmeticChecker.action"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)
```

5장 토픽, 서비스, 액션 인터페이스

5.5 빌드하기

```
$ cw
$ cbp msg_srv_action_interface_example
```

- 빌드한 후 문제가 없다면 ~/robot_ws/install/msg_srv_action_interface_example 폴더 안에 우리가 작성한 ROS 인터페이스를 사용하기 위한 파일들이 저장되어 있다.