Week6

21. ROS 2의 시간

다수의 센서를 사용하는 로봇은 시간에 따른 각 센서 값의 변화량과 그 센서들 간의 시간 동기화가 매우 중요하다. ROS 2에서는 퍼블리시 되는 토픽에 주요 데이터뿐만 아니라 해당 토픽이 퍼블리시 되는 시간을 함께 포함시킬 수 있도록 한다. stamp. frame id 를 포함하고 있는 std_msgs/msg/header 데이터 타입이 바로 그것이다.

```
$ ros2 interface show std_msgs/msg/Header.msg
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.

# Two-integer timestamp that is expressed as seconds and nanoseconds.
builtin_interfaces/Time stamp

# Transform frame with which this data is associated.
string frame_id
```

ROS 2에서 사용하는 기본 시계는 System Clock이며 rclcpp에서는 std::chrono 라이브러리를 rclpy는 time 모듈을 캡슐화하여 사용되고 있다. System Clock은 국제 표준시인 협정 세계시(UTC)로 표시되어 전 세계 어디서든 사용이 가능하다.

```
# time_rclcpp_example 패키지 미설치시
$ cd ~/ros2_ws/src
$ git clone https://github.com/robotpilot/ros2-seminar-examples.git
$ cd ~/ros2_ws && colcon build --symlink-install

$ ros2 run time_rclcpp_example time_example --ros-args -p use_sim_time:=False
[INFO] [1692150727.169510053] [time_example_node]: sec 1692150727.169509 nsec 1692150727169509110
[INFO] [1692150728.169705955] [time_example_node]: sec 1692150728.169701 nsec 1692150728169700757
[INFO] [1692150729.169713698] [time_example_node]: sec 1692150729.169708 nsec 1692150729169708332
[INFO] [1692150730.169597804] [time_example_node]: sec 1692150730.169593 nsec 1692150730169592672
[INFO] [1692150731.169704893] [time_example_node]: sec 1692150731.169700 nsec 1692150731169699666
[INFO] [1692150732.169721778] [time_example_node]: sec 1692150732.169717 nsec 1692150732169716440
```

시간 추상화(Time Abstractions)

ROS 2에서는 기본 시계 이외에도 타임머신 처럼 동작하는 시계도 사용가능하다. 이 시계는 ros2bag 나 로봇 시뮬레이션(gazebo, ignition)에서 사용할 수 있고, 이를 통해 사용자는 개발된 알고리즘을 보다 효율적으로 디버깅할 수 있다.

이를 위해 ROS 2에서는 시간을 추상화하였고 3가지의 시간을 제공하고 있다. 시계는 시간 소스(Time Source)를 통해 선언할 수 있다.

1. System Time

   System Clock를 사용한 시간이다. 이는 단조 증가하지만 타임서버와의 동기화를 통해 시간이 거꾸로 가는 경우도 있다. 예를 들면 server pc와 remote pc 간의 데이터 통신을 원활히 하기위해서는 시간을 동기화 시켜야 하는데, 다음 명령어를 각 pc에 입력하여 특정 서버 시간으로 동기화 할 수 있다.

   ```
   # ntpdate 미설치시
   $ sudo apt install ntpdate
   ```

```
$ sudo ntpdate ntp.ubuntu.com
16 Aug 10:37:17 ntpdate[7860]: adjust time server 91.189.91.157 offset 0.001216 sec
```

1. ROS Time

   보통 시뮬레이션 환경에서 시간을 조절하기 위해 많이 사용한다. 노드가 생성되기 전에 노드가 기본으로 가지고 있는 파라미터 중의 하나인 use_sim_time을 통해 사용할 수 있으며, use_sim_time이 Time로 설정된 노드는 /clock 토픽을 서브스크라이브할 때까지 시간을 0으로 초기화한다.

```
$ ros2 run time_rclcpp_example time_example --ros-args -p use_sim_time:=True
[INFO] [1692150701.447401403] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150702.447603928] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150703.447612522] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150704.447517649] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150705.447603938] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150706.447612268] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150707.447608299] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150708.447625872] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150709.447606871] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150710.447507638] [time_example_node]: sec 0.000000 nsec 0
[INFO] [1692150711.447503697] [time_example_node]: sec 0.000000 nsec 0
```

3. Steady Time

   Hardware timeouts를 사용한 시간을 말한다. 위에서 알아본 시간들과 달리 무조건 단조증가한다는 특성을 가진다.

Time API

ROS 2에서 제공하는 시간에 관련된 API는 크게 time, duration, rate가 있다.

1. Time

   Time 클래스는 시간을 다룰 수 있는 오퍼레이터를 제공하며 그 결과를 seconds 혹은 nanoseconds 단위로 변환해준다. seconds는 double형이고 nanosecond는 unsigned int의 64비트형을 가지고 nanoseconds 가 더 정확한 시간을 반환한다.

   ROS 2에서는 now멤버 함수를 통해 노드 시간을 확인할 수 있다.

```
# time_rclcpp_example/src/main.cpp 에서
rclcpp::Time now = node->now();
```

2. Duration

   Duration 클래스는 순간의 시간(timestamp)이 아닌 기간(ex)3시간 후, 1시간 전)을 다룰 수 있는 오퍼레이터를 제공하며 그 결과를 seconds/nanoseconds 단위로 반환해준다. Duration은 이전 시간을 표현할 수 있고 이는 음수로 표기된다.

```
# time_rclcpp_example/src/main.cpp 에서
rclcpp::Duration duration(1,0);
msg.stamp = now + duration;
time_publisher->publish(msg);
```

3. Rate

Rate 클래스는 반복문에서 특정 주기를 유지시켜준다. 하지만 ROS 2에서는 비슷한 기능을 하는 콜백 함수를 사용하는 Timer API를 제공하기에 이를 사용하는 것을 추천한다.

22. ROS 2의 파일 시스템

ROS 2의 기본적인 폴더 및 파일 구성을 알아보자.

패키지와 메타패키지

ROS 2에서 소프트웨어 구성을 위한 기본단위는 패키지로써 ROS 의 응용 프로그램은 패키지 단위로 개발되고 관리된다. 이러한 패키지는 공통된 목적을 지닌 패키지들을 모아둔 패키지의 집합 단위인 메타패키지(Metapackage)로 관리되기도 한다. 예를 들어 Navigation2 메타패키지는 nav2_amcl, nav2_bt_navigator 등 20여 개의 패키지로 구성되어 있다.

바이너리 설치와 소스코드 설치

ROS 패키지 설치는 바이너리 형태로 제공되어 별도의 빌드과정 없이 바로 실행하는 방법과 해당 소스코드를 직접 내려받은 후 사용자가 빌드해 사용하는 방법이 있다. 만약에 사용자가 패키지를 수정할 필요가 있다면 후자의 방법을 사용하면 된다.

예를 들어 teleop_twist_joy 패키지를 설치할 때 2가지 방법으로 설치할 수 있다.

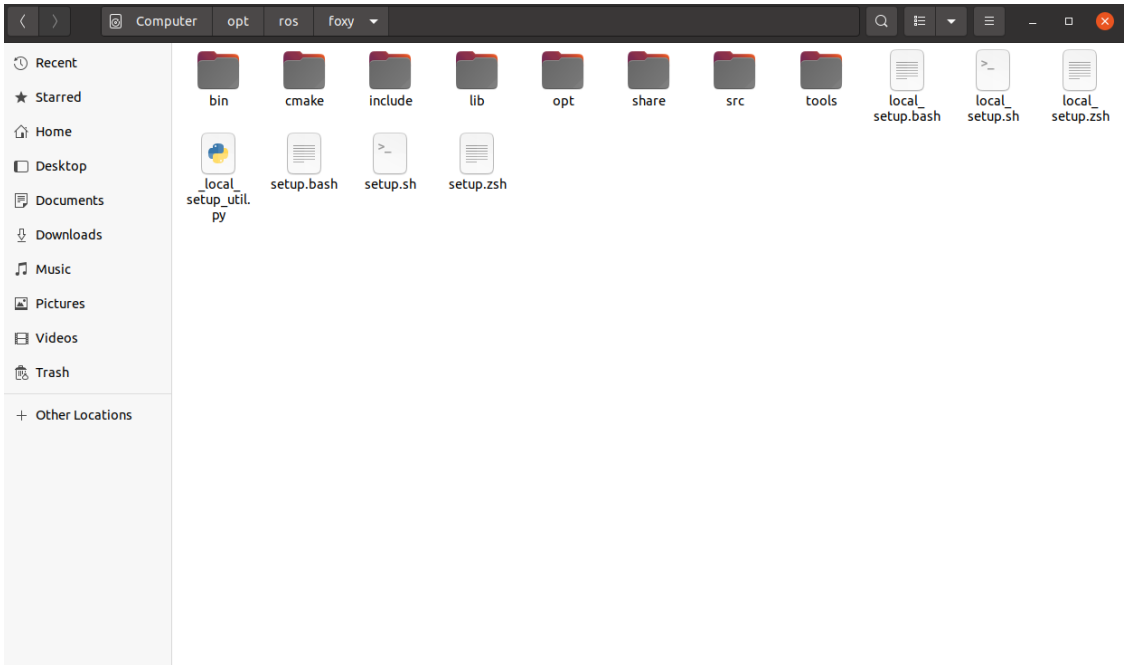
```
# 바이너리 설치
$ sudo apt install ros-foxy-teleop-twist-joy

# 소스코드 설치
$ cd ~/ros2_ws/src
$ git clone https://github.com/ros2/teleop_twist_joy.git
$ colcon build --symlink-install --package-select teleop_twist_joy
```

기본 설치 폴더와 사용자 작업 폴더

ROS 2는 /opt/ros/[버전 이름] 폴더에 설치된다. 예를 들어 ROS 2 Foxy Fitzroy 버전을 설치하면 ROS 가 설치된 경로는 다음과 같다.

- 기본 설치 경로: /opt/ros/foxy



사용자 작업 폴더는 사용자가 원하는 곳에 생성할 수 있다. 나는 ~/ros2_ws/ 를 사용한다. 즉 /home/ros2_ws를 사용한다.

23. ROS 2의 빌드 시스템과 빌드 툴

빌드 시스템(Build System)과 빌드 툴(Build Tools)을 나누어 비교하고 알아보자. 빌드 시스템은 단일 패키지를 대상으로 하며, 빌드 툴은 시스템 전체를 대상으로 한다.

빌드 시스템

ROS 2에서는 ament라는 빌드 시스템을 사용한다. ament중에서 많이 쓰이는 ament_cmake는 ROS 1에서 사용되는 빌드 시스템인 catkin의 업그레이드 버전으로 Cmake의 빌드 설정 파일인 CmakeList.txt에 기술된 빌드 설정을 기반으로 빌드를 수행하게 된다. 또한 ament_python을 사용하여 Cmake를 사용하지 않는 파이썬 패키지 관리도 지원한다. 파이썬 패키지들은 setup.py 파일에 기술된 빌드 설정을 기반으로 빌드를 수행하게 된다.

빌드 툴

ROS 2에서는 빌드 툴로서 colcon을 추천한다. 사용자 작업 폴더에서 colcon build라는 명령어에 다양한 옵션과 함께 사용하여 빌드하게 된다.

ROS 2 패키지를 생성하기 위해서는 2가지 방법이 있는데, 하나는 직접 패키지 폴더를 만들고 그안에 파일 시스템에 필수적인 package.xml이나 CmakeLists.txt 또는 setup.py 등을 포함시켜 주고 소스코드를 작성하는 것과 ros2cli 명령어를 이용하는 것이다.

패키지 생성

패키지 생성 명령어는 다음과 같다. ros2 pkg create 명령어를 사용하고 그뒤에 옵션을 붙여주면 된다.

```
$ ros2 pkg create [패키지 이름] --bulid-type [빌드 타입] --dependencies [의존하는패키지1] [의존자는패키지n]
```

빌드 타입은 RCL로 C++을 사용한다면 ament_cmake를 설정하고, python이면 ament_python을 사용한다. 참고로 GUI 프로그램을 작성해야 한다면 rqt plugin 계열을 써야하기 때문에 python이여도 ament_cmake를 사용한다.

```
$ ros2 pkg create my_first_ros_rclcpp_pkg --build_type ament_cmake --dependencies rclcpp std_msgs
```

```
$ ros2 pkg create my_first_ros_rclpy_pkg --build_type ament_python --dependencies rclpy std_msgs
```

→ std_msgs(ROS의 표준 메시지패키지)와 rclpy(ROS에서 파이썬 사용하기위한 라이브러리)

dependencies(의존하는 패키지)설정은 패키지 생성할 때 지정할 수도 있지만, 생성한 다음 package.xml에서 직접입력해도된다.

패키지 빌드

패키지를 빌드할 때에는 colcon 빌드 툴을 사용한다. workspace 폴더로 이동하고 colcon build 명령어로 전체를 빌드하게 된다. 여기서 빌드 옵션을 추가하여 사용하게 된다.

symlink를 사용하게 된다면 —symlink-install 옵션을 사용하고

패키지를 선택하여 빌드할 때는 —packages-select, 특정 패키지의 의존성 패키지들까지도 함께 빌드하게 하려면 —packages-up-to 옵션을 사용하게 된다.

```
$ cd ~/ros2_ws && colcon build --symlink-install --package-select [패키지 이름]
```

빌드 시스템에 필요한 부가 기능

- vcstool(버전 컨트롤 시스템 툴)

  vcs는 Version Control System의 약자로 ROS 커뮤니티에서 사용하는 vcstool의 실행 명령어이다.

- rosdep(의존성 관리 툴)

  rosdep은 package.xml에 기술된 의존성 정보를 가지고 의존성 패키지들을 설치해 주는 역할을 하게 된다.

24. ROS 2의 패키지 파일

패키지 파일은 패키지 설정 파일 package.xml, 빌드 설정 파일 CMakeLists.txt, 파이썬 패키지 설정 파일 setup.py, 파이썬 환경설정 파일 setup.cfg, RQt 플러그인 설정 파일 plugin.xml, 패키지 변경로그 파일 CHANGELOG.rst, 라이선스 파일 LICENSE, 패키지 설정 파일 README.md가 있다.

패키지 설정 파일 (package.xml)

패키지 설정 파일은 ROS 패키지의 필수 구성 요소로서 패키지의 정보를 기술하는 파일이다.

패키지 이름, 저작자, 라이선스, 의존성 패키지 등의 내용을 포함

사용되는 빌드 툴, 의존서 패키지들이 모두 기술되기에 빌드 및 패키지 설치, 사용에 있어 매우 중요하다.

모든 ROS 패키지의 필수파일로 각 패키지당 무조건 1개의 package.xml 포함한다

```
<?xml version="1.0"?> # 문서문법 정의 문구
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">  # 패키지 설정 파일 버전 3(Ros2사용버전)
  <name>bridge_python</name> # 패키지 이름
  <version>0.0.0</version> # 패키지 버전
  <description>TODO: Package description</description> # 패키지의 간단한 설명
  <maintainer email="rodel@todo.todo">rodel</maintainer> #패키지 관리자이름 및 이메일
  <license>TODO: License declaration</license> # 라이선스 기재.

  <test_depend>ament_copyright</test_depend> # 패키지 테스트 시 필요한 의존 패키지 이름
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <exec_depend>rclpy</exec_depend> # 패키지 실행 시 필요한 의존 패키지 이름
  <exec_depend>std_msgs</exec_depend>

  <export> # 위에서 명시하지 않은 확장 태그명 사용할 때 쓴다. <build_type> <rviz> <rqt_gui> <deprecated> 등의 태그가 있따
    <build_type>ament_python</build_type>
  </export>
</package>
```

빌드 설정 파일 (CMakeLists.txt)

Ros2의 빌드 시스템인 ament는 CMakeLists.txt에 빌드 환경을 기술해 사용한다. 실행 파일 생성, 의존성 패키지 우선 빌드, 링크 생성 등을 설정한다.

CMake 사용이유는 Ros 패키지를 멀티 플랫폼에서 빌드할수 있게끔 하기 위해서다.

rclpp가 아닌 rclpy 패키지는 순수 파이썬이라 CMakeLists.txt 는 없다 → 대신 setup.py 사용함

파이썬 패키지 설정 파일 (setup.py)

순수 Ros2 파이썬 패키지에서만 사용하는 파일. CMakeLists.txt와 package.xml의 기능을 한다.

package.xml은 ros패키지의 필수 구성 요소이기에 비슷한 내용을 기입하더라도 패키지에 포함시켜야 한다.

```
from setuptools import setup

package_name = 'bridge_python'
submodules = 'bridge_python/submodules'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name], # 의존하는 패키지, 하나씩 나열해도 되지만
                             # find_packages()를 기입해주면 자동으로 의존하는 패키지를 찾아준다.
    data_files=[ # 패키지에서 사용하는 파일들 기입해 함께 배포함
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],# 의존하는 패키지.
    zip_safe=True,
    maintainer='rodel',
    maintainer_email='rodel@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={ # 플랫폼별로 콘솔 스크립트를 설치하도록 콘솔 스크립트 이름과 호출 함수를 기입한다
        'console_scripts': [
        'practice_CAN_node = bridge_python.practice_CAN_node:main',
        'practice_control_anchor_node = bridge_python.practice_control_anchor_node:main',
        ],
    },
)
```

파이썬 환경설정 파일 (setup.cfg)

setup.py 함수에서 설정하지 ahtgksms rlxk dhqtusdmf wjddmlgkf tn dlTek.

develop, install옵션을 설정해 스크립트의 저장위치를 설정한다.

RQt 플러그인 설정 파일 (plugin.xml)

패키지 변경로그 파일 (CHANGELOG.rst)

패키지의 업데이트 내역을 기술하는 파일

라이선스 파일 (LICENSE)

코드에 사용한 라이선스를 기술하는 파일

패키지 설정 파일 (README.md)

패키지의 부가 설명을 기술하는 파일. 필수파일은 아니지만 사용자를 배려하는 문서다.