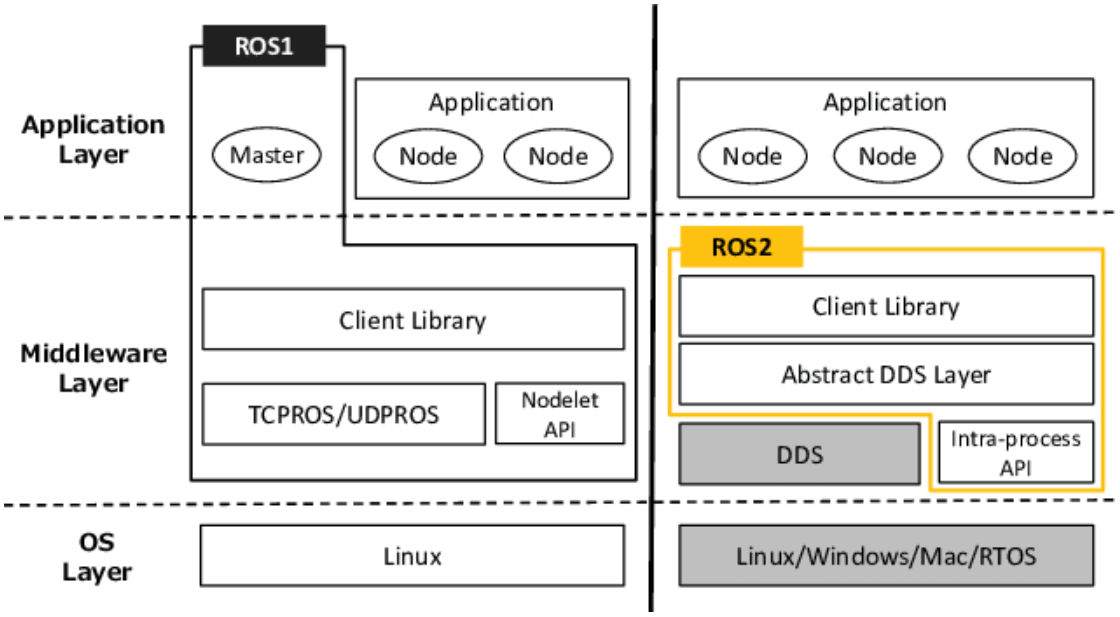
ROS 프로그래밍: Package - Python

해당 실습 자료는 표윤석, 임태훈 “ROS2로 시작하는 로봇 프로그래밍” 루비페이퍼 (2022)를 참고하여 제작하였습니다.

📌목차

ROS2 DDS와 QoS

DDS(Data Distribution Service)



- ROS2는 산업용 시장을 위해 표준 방식 사용을 중요하게 여겼고, ROS 1때와 같이 자체적으로 만들기보다는 산업용 표준을 만들고 생태계를 꾸려가고 있었던 DDS를 통신 미들웨어로써 사용하였습니다.

- 특징
  - **산업 표준**: 비영리단체인 OMG(Object Management Group)가 분산 객체에 대한 기술 표준을 정하고자 만든 미들웨어로 **산업표준**으로 자리잡았습니다.
  - **운영체제 독립**: DDS는 Window,Linux, MacOS, Android등 다양한 운영체재를 지원합니다.
  - **언어 독립:** DDS는 미들웨어로 그 상위 레벨인 사용자 코드 레벨에서 프로그래밍 언어를 바꾸지 않아도 됩니다. ROS2에서도 RMW(Ros Middle Ware)로 디자인되어 다양한 언어를 지원합니다.
  - **UDP기반의 전송 방식**: 여러 목적지로 동시에 보낼 수 있다는 장점이 있으나 TCP방식에 비해 신뢰성이 떨어진다는 단점이 있었습니다. 이는 QoS를 도입하여 어느정도 해결되었습니다.
  - **데이터 중심적**: DDS 사양 중 DCPS(Data-Centric Publish-Subscribe)는 적절한 수신자에게 적절한 정보를 효율적으로 전달하는 것을 목표로 하는 발간 및 구독 방식입니다.
  - **동적 검색:** DDS는 동적검색을 제공하여 어떤 토픽이 지정 도메인 영역에 있고 어떤 노드가 이를 발산하고 수신하는 지 알 수 있습니다. ROS1의 노드들 이름 지정 및 등록, 노드간의 메세지 연결 등을 지원하던 ROS Master를 사용하지 않고 노드들이 Participant로 취급되어 동적 검색을 통해 연결할 수 있게 되었습니다.

- 실시간 데이터 전송을 보장하였고, DDS의 사용으로 노드 간의 동적 검색 기능을 지원하고 있어서 기존 ROS 1에서 각 노드들의 정보를 관리하였던 ROS Master가 없어도 여러 DDS 프로그램 간에 통신할 수 있습니다.

  ⇒ DDS를 사용하므로써 퍼블리시, 서브스크라이브형 메시지 전달이 가능해지고, 실시간 데이터 전송, 불안정한 네트워크에 대한 대응, 보안 등이 강화되었다

QoS (Quality of Service)

- DDS의 서비스 품질 (QoS, Quality of Service)로, 데이터 통신의 옵션으로 생각하시면 됩니다.
- ROS 2에서는 TCP처럼 신뢰성을 중시여기는 통신 방식과 UDP처럼 통신 속도에 포커스를 맞춘 통신 방식을 선택적으로 사용할 수 있습니다.
- 이를 위해 DDS의 QoS를 도입하였고 퍼블리셔나 서브스크라이브 등을 선언할 때 QoS를 매개 변수 형태로 지정할 수 있어서 원하는 데이터 통신의 옵션 설정을 유저가 직접할 수 있게 되어 있습니다.
- **QoS 종류**
  - ROS 2에서는 TCP처럼 데이터 손실을 방지함으로써 신뢰도를 우선시하거나 (reliable), UDP처럼 통신 속도를 최우선시하여 사용(best effort)할 수 있게 하는 **Reliability**
  - 통신 상태에 따라 정해진 사이즈만큼의 데이터를 보관하는 **History**
  - 데이터를 수신하는 서브스크라이버가 생성되기 전의 데이터를 사용할지 폐기할지에 대한 설정인 **Durability**
  - 정해진 주기 안에 데이터가 발신 및 수신되지 않을 경우 이벤트 함수를 실행시키는 **Deadline**
  - 정해진 주기 안에서 수신되는 데이터만 유효 판정하고 그렇지 않은 데이터는 삭제하는 **Lifespan**
  - 정해진 주기 안에서 노드 혹은 토픽의 생사 확인하는 **Liveliness**

  ▼ QoS 종류와 종류별 Value

  - History

History	**데이터를 몇 개나 보관할지를 결정하는 QoS 옵션**
KEEP_LAST	정해진 메시지 큐 사이즈 만큼의 데이터를 보관 *** depth**: 메시지 큐의 사이즈 (KEEP_LAST 설정일 경우에만 유효)
KEEP_ALL	모든 데이터를 보관 (메시지 큐의 사이즈는 DDS 벤더마다 다름)

  - Reliability

Reliability	**데이터 전송에 있어 속도를 우선시 하는지 신뢰성을 우선시 하는지를 결정하는 QoS 옵션**
BEST_EFFORT	데이터 송신에 집중. 전송 속도를 중시하며 네트워크 상태에 따라 유실이 발생할 수 있음
RELIABLE	데이터 수신에 집중. 신뢰성을 중시하며 유실이 발생하면 재전송을 통해 수신을 보장함

  - Durability

Durability	**데이터를 수신하는 서브스크라이버가 생성되기 전의 데이터를 사용할지 폐기할지에 대한 QoS 옵션**
TRANSIENT_LOCAL	Subscription이 생성되기 전의 데이터도 보관 (Publisher에만 적용 가능)
VOLATILE	Subscription이 생성되기 전의 데이터는 무효

  - Deadline

Deadline	**정해진 주기 안에 데이터가 발신 및 수신되지 않을 경우 EventCallback를 실행시키는 QoS 옵션**
deadline_duration	Deadline을 확인하는 주기

- Lifespan

Lifespan	**정해진 주기 안에서 수신되는 데이터만 유효 판정하고 그렇지 않은 데이터는 삭제하는 QoS 옵션**
lifespan_duration	Lifespan을 확인하는 주기

- Liveliness

Liveliness	**정해진 주기 안에서 노드 혹은 토픽의 생사 확인하는 QoS 옵션**
liveliness	자동 또는 매뉴얼로 확인할지를 지정하는 옵션, 하기 3가지 중 선택 * AUTOMATIC, MANUAL_BY_NODE, MANUAL_BY_TOPIC 중 선택
lease_duration	Liveliness을 확인하는 주기[출처] 019 DDS의 QoS(Quality of Service) (오픈소스 소프트웨어 & 하드웨어: 로봇 기술 공유 카페 (오로카)) | 작성자 표윤석

ROS 패키지

- 소프트웨어 구성을 위한 기본 단위로, ROS의 응용프로그램은 패키지 단위로 개발되고 관리됩니다.
- 노드를 하나 이상 포함하거나, 다른 노드를 실행하기 위한 런치(launch)와 같은 실행 및 설정 파일들을 포함합니다.

바이너리 설치와 소스 코드 설치

ROS 패키지 설치 방식은 2가지가 있습니다.

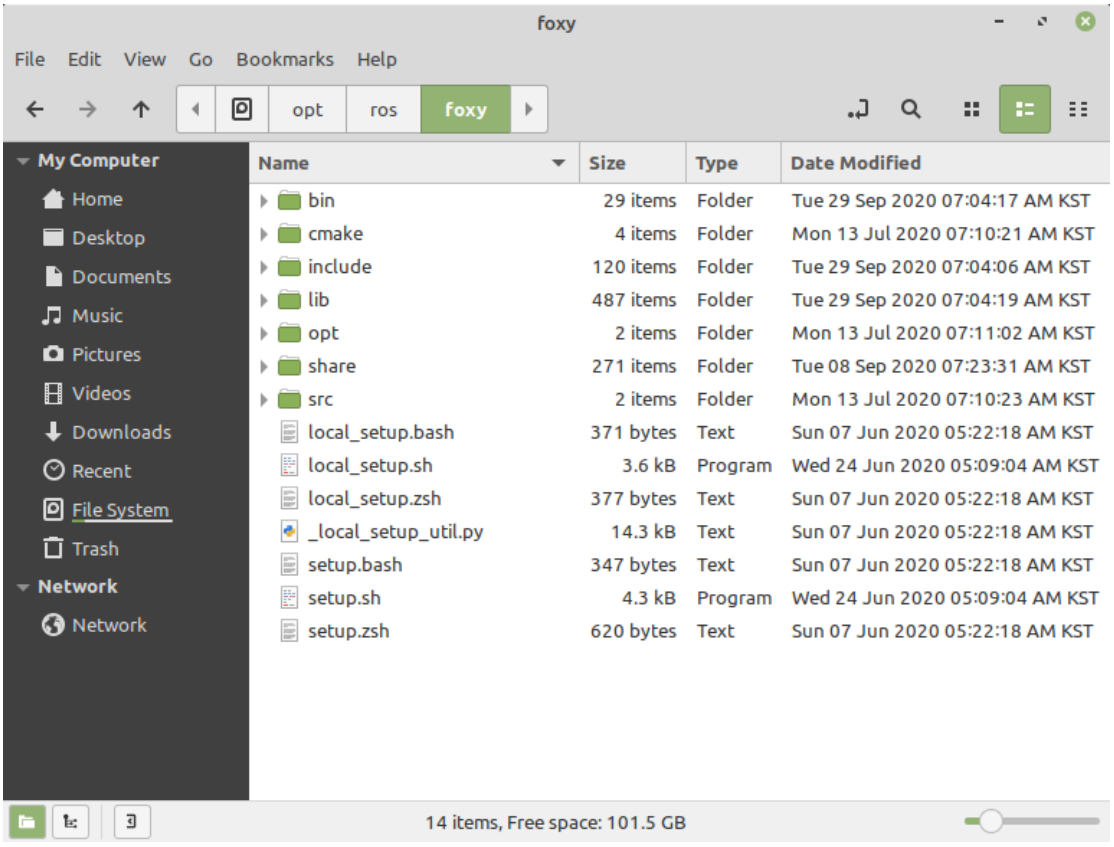
1. **바이너리 설치**
   - 별도의 빌드 없이 실행 가능합니다.
   - `/opt/ros/foxy`에 저장되어 `ros2 run`이나 `ros2 launch`로 해당 패키지 내의 노드를 실행할 수 있습니다.
   - ex) `sudo apt install ros-foxy-teleop-twist-joy`
2. **소스코드 설치**
   - 소스 코드를 직접 다운로드 한 후 사용자가 빌드하여 사용하는 방법입니다.
   - 패키지를 수정하거나 소스 코드 내용을 확인할 필요가 있을 때 사용하는 방식입니다.
   - 사용자 작업폴더(예: `~/ros2_ws/src`)에 `git clone` 명령어를 통해 원격의 레포지토리를 복사한 뒤 직접 빌드하는 방식입니다.

- ex)

```
$ cd ~/ros2_ws/src
$ git clone https://github.com/ros2/teleop_twist_joy.git
$ cd ~/ros2_ws/
$ colcon build --symlink-install \
--packages-select teleop_twist_joy
```

기본 설치 폴더와 사용자 작업 폴더

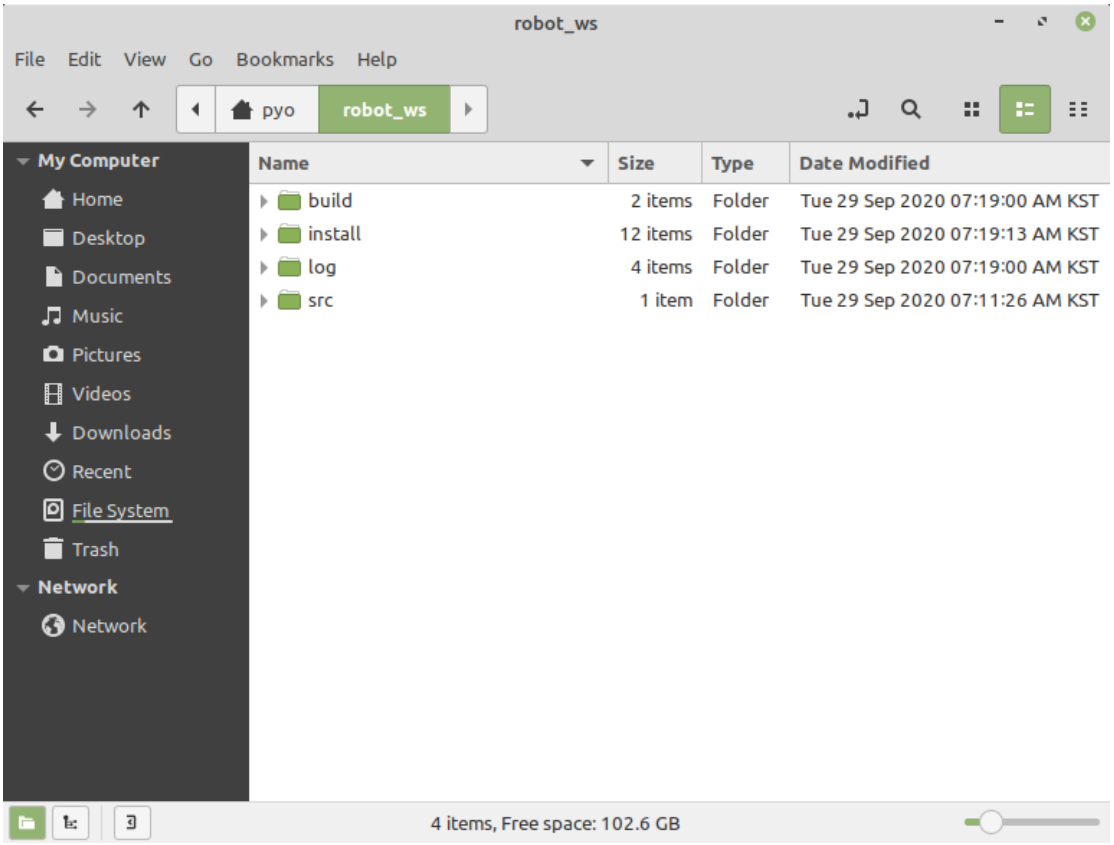
- **설치 폴더**
  - `/opt/ros/{ros2버전}` (예: `/opt/ros/foxy`)



  - 세부 내용
    - /bin:실행 가능한 바이너리 파일
    - /cmake: 빌드 설정 파일
    - /include: 헤더 파일
    - /lib: 라이브러리 파일
    - /opt: 기타 의존 패키지

    - /share: 패키지의 빌드, 환경 설정 파일
    - local_setup.*: 환경 설정 파일*
    - *setup.*: 환경 설정 파일

- **사용자 작업 폴더**
    - 사용자가 원하는 곳에 폴더명으로 생성할 수 있습니다. (예: `~/ros2_ws/src`)



    - 세부 내용
        - /build: 빌드 설정 파일용 폴더
        - /install: msg, srv 헤더 파일과 사용자 패키지 라이브러리, 실행 파일용 폴더
        - /log: 빌드 로깅 파일용 폴더
        - /src: 사용자 패키지용 폴더

빌드 시스템과 빌드 툴

빌드 시스템(build system)

- **단일 패키지**의 의존성을 해결하고 빌드하여 실행 가능한 파일을 생성하는 것입니다.

- ROS2는 새로운 빌드 시스템인 ament를 이용하며, ament_cmake는 ROS1에서 사용하던 catkin의 업그레이드 버전이고, ament_python을 통해 CMake를 사용하지 않는 Python 패키지 관리가 가능해졌습니다.
    - ROS1의 빌드 시스템은 CMake만 지원하였으며, Python은 CMake 내의 사용자 정의 로직을 이용하여 처리하였습니다.

빌드 툴(build tool)

- ROS는 **여러 패키지**를 함께 빌드하는 구조이므로, 패키지별로 서로 다른 빌드 시스템을 호출하고 패키지들의 종속성이 얽혀있는 경우, 이를 풀고 토폴로지 순서대로 빌드해야 합니다. 이를 위해 사용하는 것이 빌드 툴입니다.
- ROS 빌드 툴은 각 패키지에 기술되어 있는 종속성 그래프를 해석하고 토폴로지 순서로 각 패키지에 대한 특정 빌드 시스템을 호출합니다.
- ROS 빌드 툴로는 rosbuild, catkin_make, catkin_make_isolated, catkin_tools, ament_tools 그리고 현재 ROS 2 버전에서 널리 사용되고 있는 **colcon**이 있습니다.
- **colcon**은 CLI 형태의 명령어로 사용하며, ROS 1과 ROS 2 모두를 지원하기 위하여 통합된 빌드 툴로서 소개되었으며 ROS 2 Bouncy 이후 ROS 2의 기본 빌드 툴로 사용중입니다.

패키지 생성

▼ 리눅스 명령어

- `cd` : 경로 이동
- `mkdir` : 디렉토리 생성
- `source` : 설정 파일 변경사항 반형을 위한 명령어
- `rm` : 삭제 (-r 옵션을 추가할 경우 디렉토리를 삭제할 수 있음.)
    - ex) `rm helloworld.py` `rm -r ./helloworld`
- `pwd` : 현재 경로 확인 가능
- `ls` : 디렉토리에 있는 파일이나 디렉토리를 확인할 수 있음
    - 옵션 정보
        - -a : 숨긴 파일도 포함하여 출력
        - -l : 파일의 상세 정보 출력
        - -F: 파일의 종류를 표시(*:실행파일, /: 디렉토리 @: 심볼릭 링크)
- `clear` : 화면을 깨끗하게 지우는 명령어
- `exit` : 명령어 창에서 작업을 마치고 창을 닫을 때 쓰는 명령
- `mv` : 파일 및 디렉토리 이동
    - `mv {옮길 파일/디렉토리 경로} {옮기고싶은 목적 경로}`

패키지 생성

- ROS 2 패키지 생성 명령어는 다음과 같다. `` `ros2 pkg create` `` 명령어를 사용하고 그 뒤에 옵션을 붙여 사용합니다.

```
$ mkdir ros2_ws
$ cd ros2_ws
$ mkdir src
$ cd src
$ source /opt/ros/foxy/setup.bash

# ros2 pkg create [패키지이름] --build-type [빌드 타입]
#                       --dependencies [의존하는패키지1] [의존하는패키지n]

$ ros2 pkg create topic_helloworld --build-type ament_python \
$        --dependencies rclpy std_msgs
```

- 해당 실습은 파이썬을 이용하기 때문에 클라이언트 라이브러리 `rclpy` 와 ROS의 표준 메세지 패키지인 `std_msgs` 를 필요로 하며, 만약 해당 패키지들이 미리 설치가 진행되어있지 않았다면 사전에 설치를 진행해야만 합니다.
  - dependencies 관련
    - **std_msgs**: ROS의 표준 메시지 패키지 std_msgs를 사용하겠다는 의미
    - **rclpy**: 파이썬을 사용하기 위한 클라이언트라이브러리 rclpy를 사용하겠다는 의미
- 패키지 생성이 끝난 경우 패키지가 갖추어야 할 기본 내부 폴더 그리고 package.xml 파일들이 다음과 같이 생성 됩니다.

```
.
├── topic_helloworld
│   └── __init__.py
│
├── resource
│   └── topic_helloworld
├── test
│   ├── test_copyright.py
│   ├── test_flake8.py
│   └── test_pep257.py
├── package.xml # 패키지 설정 파일
├── setup.cfg # 파이썬 패키지 환경설정 파일
└── setup.py # 파이썬 패키지 설정 파일

3 directories, 8 files
```

패키지 설정

- 앞서 생성한 기본 파일 중 `package.xml`, `setup.cfg`, `setup.py`은 환경 설정과 빌드 설정에 필요한 패키지 파일에 해당합니다.
- 해당 파트에 대한 자세히 설명은 자세히 다루지 않습니다. 자세한 내용이 궁금하신 분들은 아래의 링크를 통해 확인하시길 바랍니다.

Creating your first ROS 2 package - ROS 2 Documentation: Foxy documentation

A package can be considered a container for your ROS 2 code. If you want to be able to install your code or share it with others, then you'll need it organized in a package. With packages, you can release your ROS 2 work and allow others to build and use it easily.

2 https://docs.ros.org/en/foxy/Tutorials/Creating-Your-First-ROS2-Package.html

패키지 설정 파일(package.xml)

- ROS 패키지의 필수 구성 요소로서 패키지의 정보를 XML 형태로 기술한 파일
- 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 작성

▼ package.xml

```
<?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>topic_detection</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="jetson@todo.todo">jetson</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>std_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>
```

▼ 관련 요소

- **<test_depend>**: 패키지를 테스트할 때 필요한 의존 패키지 이름
- **<export>**: 위에서 명시하지 않은 확장 태그명을 사용할 때 이용
  - **<build_type>**: 빌드 타입
  - **<rviz>**: RViz 플러그인에 사용
  - **<rqt_guyi>**: RQt 플러그인에 사용
  - **<deprecated>**: 더이상 사용되지 않게 되는 경우 사용자에게 알려줄 수 있는 태그

▼ 추가 문법

```
<?xml>: 문서 문법 정의
<package> </package>: ROS 패키지 설정 부분. format="3"은 패키지 설정
파일 버전이 3이라는 뜻.
<name>: 패키지 이름
<version>: 패키지 버전
<description>: 패키지 설명
<maintainer>: 패키지 관리자의 이름과 메일주소
<license>
<url>: 패키지를 설명하는 웹 페이지/소스코드 저장소 등의 주소 기재
<author>: 패키지 개발에 차여한 개발자의 이름과 메일주소
<buildtool_depend>
<build_depend>: 패키지 빌드 시 필요한 의존 패키지 이름
<exec_depend>: 패키지 실행 시 필요한 의존 패키지 이름
<test_depend>: 패키지 테스트 시 필요한 의존 패키지 이름
<export>: 위에 없는 태그명을 여기에 다 씀. <build_type>,<rviz>,<rqt_
gui>,<deprecated>등
```

파이썬 패키지 설정 파일(setup.py)

- setuptools을 이용하여 다양한 배포를 위한 설정을 해주는 파일

▼ setup.py

```
from setuptools import setup

package_name = 'topic_detection'

```

```
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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='jetson',
    maintainer_email='jetson@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```

▼ 관련 요소

- **packages** : 의존 패키지
  - 의존 패키지 리스트를 나열해도 됨
  - **find_packages()**를 이용할 경우 의존 패키지를 자동으로 찾아줌

    `from setuptools import find_packages`
- **data_files** : 이 패키지에서 사용되는 파일들 기입하여 함께 배포
  - ROS에서는 주로 resource 폴더 내의 ament_index를 위한 패키지의 이름의 빈 파일이나 package.xml, *.launch.py, *.yaml등을 기입
- **install_requires :** 의존하는 패키지. 이 패키지를 pip로 설치할 때, 해당 패키지들도 함께 설치함
- zip_safe: 설치시 zip으로 아카이브 할 지 여부 결정
- **tests_require** : 테스트에 필요한 패키지. ROS는 pytest사용
- **classifiers** : PyPI에 등록될 메타 데이터 설정으로 PyPI페이지 좌측 Meta란에서 확인 가능
- **description** : 패키지 설명
- **entry_points**: 플랫폼 별로 콘솔 스크립트를 설치하도록 콘솔 스크립트 이름과 호출 함수를 기입

- console_scripts키를 사한 실행 파일 설정

  ex) hello_world

```
entry_points={
    'console_scripts': [
        'helloworld_publisher = my_first_ros_rclpy_pkg.helloworld_publisher:main',
        'helloworld_subscriber = my_first_ros_rclpy_pkg.helloworld_subscriber:main',
    ],
},
```

  my_first_ros_rclpy_pkg.helloworld_publisher 모듈 `(경로/puslisher의 *.py이름)`과my_first_ros_rclpy_pkg.helloworld_subscriber 모듈 `(경로/subscriber의 *.py이름)`의 main함수를 호출

파이썬 패키지 환경설정 파일(setup.cfg)

- 배포를 위한 구성 파일
- setup.py의 setup함수에서 설정하지 못하는 기타 옵션을 정의

▼ setup.cfg

```
[develop]
script-dir=$base/lib/topic_helloworld
[install]
install-scripts=$base/lib/topic_helloworld
```

▼ 관련 요소

아래의 옵션을 통해 **스크립트의 저장 위치 설정**

- **[develop]**
- **[install]**

빌드

1. `cd ..{root 경로}/` src 밖으로 이동
2. `colcon build` : 패키지 빌드

   추가 옵션:

- (✅) `--symlink-install` : 빌드시 관련 파일을 링크로 연결하므로 쉽게 수정이 가능.(빌드 후 python을 수정해도 재빌드 하지 않아도 됨)
- `--packages-select [패키지 이름]` : 특정 패키지만 빌드
- `--packages-up-to [패키지 이름]` : 특정 패키지의 의존성 패키지도 함께 빌드

실행

- `ros2 run {패키지 이름} {entry_points에 설정한 key값}`

```
##  터미널 1
#============ 터미널 실행시 항상 실행 ============
$ source /opt/ros/foxy/setup.bash
$ source ./install/local_setup.bash
#================================================

# ros2 run [패키지 이름] [entry_point에 설정한 key값]
$ ros2 run topic_helloworld helloworld_publisher
```

```
##  터미널 2
#============ 터미널 실행시 항상 실행 ============
$ source /opt/ros/foxy/setup.bash
$ source ./install/local_setup.bash
#================================================

# ros2 run [패키지 이름] [entry_point에 설정한 key값]
$ ros2 run topic_helloworld helloworld_subscriber
```

💡 install 디렉토리에 위치한 `local_setup`과 `setup`은 뭐가 다른 걸까요?

- `local_setup` 은 내가 설치한 패키지의 환경 변수를 source 하기 위한 파일!
- `setup` 은 `/opt/ros/foxy` 와 같이 글로벌하게 사용되는 환경 변수도 source 합니다.
  즉,
  `source /opt/ros/foxy/setup.bash & source install/setup.bash` 과 동일합니다.