Workspace Creation & Description Package

Stage 1: Creating the Workspace and Building with Colcon

Objective

Set up a ROS2 development workspace for the mycobot630pro robotic arm project and learn how to build packages using colcon.

Prerequisites

- Ubuntu 22.04 LTS
- ROS2 Jazzy Jalisco installed
- Basic terminal and Linux knowledge

Step-by-Step Guide

1. Check ROS2 Jazzy Jalisco Installation

- Ensure ROS2 Jazzy Jalisco is installed on your system. If not, follow the official installation guide: ROS2 Jazzy Jalisco Installation Guide
- The ROS2 Distribution is installed by default in this path "/opt/ros/" and to be able to use ros2 CLI and packages, the **setup.bash** file needs to be sourced either in each terminal or sourced in (**.bashrc).**

```
# Sourcing the ROS2 Distro (for each terminal)
source /opt/ros/jazzy/setup.bash
# echo the source command in bashrc to be used for any new term:
echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
```

- To check the ros2 distro, we can either look for the content in */opt/ros/* or we can check for environment variables

```
# Check /opt/ros folder
ls /opt/ros/
# Check for environment variables
printenv | grep -i ROS
```

For more detailed information please see:

Configuring ROS2 environment

2. Create Workspace Directory

- Usually workspace directory is named after the building tool it will be using, so for most of ROS2 projects you can see that the workspace directory is named *colcon_ws* indicating that it is using *colcon* as it is building tool. And for ROS1 project it is usually *catkin_ws* , but this is just naming convention and not obligatory.

```
#we create the src folder that will contain all the packages
as well
mkdir -p ~/colcon_ws/src
```

References

- ROS2 Workspace Tutorial
- Using colcon to build packages
- Colcon Build Documentation

Stage 2: Package Creation and Workspace Building

Objective

Creating packages in ROS2 Workspace (Ex. Creating description package for the robot)

What is a ROS2 Package?

- In ROS2, a package is a fundamental organizational unit for your code. Packages allow you to structure your ROS2 work in a way that makes it easy to install, share, and reuse your code.
- ROS2 utilizes ***ament*** as its build system and ***colcon*** as its build tool.
- Packages can be created using either ***CMake*** or ***Python***, which are officially supported. Although other build types are possible, these two are the most common and well-supported.

Anatomy of a ROS2 Package

Both Python and CMake packages have specific required files and folders.

The basic structure of each type of package is as follows:

CMake Packages:

- **CMakeLists.txt**: This file contains instructions on how to build the code within the package.
- **include/<package_name>**: This directory stores public header files for the package.
- **package.xml**: This file holds meta-information about the package, such as its name, version, description, maintainer, and license.
- **src**: This directory contains the source code for the package.
- **resource/<package_name>**: A marker file for the package.

Python Packages:

- **package.xml**: Similar to CMake packages, this file holds meta-information about the package.
- **setup.py**: Contains instructions for how to install the package.
- **setup.cfg**: Required when a package has executables, allowing ros2 run to locate them.
- **<package_name> directory**: A directory with the same name as the package, which contains the __init__.py file and is used by ROS2 tools to find the

package.

- **resource/<package_name>**: A marker file for the package.

A simple package might have a file structure that resembles this:

```
my_package/
    CMakeLists.txt    (for C++ packages)
    include/my_package/
    package.xml
    src/
    my_package/          (for python packages)
    package.xml
    resource/my_package
    setup.cfg             (for python packages)
    setup.py              (for python packages)
    my_package/           (for python packages)
```

Key differences between C++ and Python Packages

Package Type	C++	Python
Build system	C++ packages use CMake	Python packages use setuptools configured by setup.py
Compilation	C++ code needs to be compiled	Python code is interpreted, so python packages do not include source files, but instead python scripts in its directory
Header files	C++ packages use header files located in the include directory	Python packages don't use header files

References

- ROS2 Creating Packages

Step-by-Step Guide: Creating the python package

Here's how to create a ROS2 Python package named mycobot630pro_description, which will house the URDF and resources for the MyCobot 630 Pro robotic arm:

1. Create Python Package

```
# Navigate to the source folder
cd ~/colcon_ws/src/
# Create the package with build type (ament_python)
ros2 pkg create --build-type ament_python mycobot630pro_description
```

This will generate a new directory named mycobot630pro_description within your src folder, including the necessary files and folders.

2. Copying Necessary URDF and mesh files

In our case, we already have the actual robot's description files from the manufacturer. So we can simply get the description files of our model and copy it into our description file.

Note: *In the case of working on a custom robot, we would be extracting the urdf file from the mechanical CAD design of the robot as well as the mesh files.*

1. Download the description files from manufacturer repo: mycobot_pro_630 description
2. Create a *urdf* folder inside our package
3. Copy and paste the files inside urdf folder

The package's file structure should resemble this:

```
mycobot630pro_description/
        mycobot630pro_description/
            __init__.py
    resource/
        mycobot630pro_description
      urdf/
          mycobot_pro_630/ (the copied folder)
              base.dae
```

```
               link1.dae
               link2.dae
               link3.dae
               link4.dae
               link5.dae
               link6.dae (to be modified)
               mycobot_pro_630.urdf
     package.xml
     setup.cfg
     setup.py (to be modified to include the urdf file in the
 data_files)
```

3. Modifying mesh files (link6.dae)

1. The desired modification of a specific mesh file can be made by using any Mechanical CAD Software (eg. *Solidworks*, *Autodesk Fusion 360*)
2. The modified file must be extracted as .dae file
3. link6.dae (after modifications) gets replaced in the *urdf/mycobot_pro_630* folder.

4. Editing package and setup files.

1. Package.xml has the package information, we can edit the description of the pacakge, the maintainer email and name, and the license. We also add dependencies but we do not need to do so in this pacakge

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_forr
<package format="3">
  <name>mycobot630pro_description</name>
  <version>0.4.0</version>
  <description>The mycobot630pro_description package</descrip
  <maintainer email="todo@todo.todo">todo</maintainer>
  <license>BSD</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>
```

2. Then we modify *setup.py* : when we have files that we want to share with other packages or be called and accessed by our pacakge, then we add these files path to *date_files* attribute of the *setup*() method as in this code snippet.

```
import os
from setuptools import setup, __version__ as setuptools_versi
from packaging.version import Version
from glob import glob # To use for the files path

package_name = 'mycobot630pro_description'

# setuptools
use_dash_separated_options = Version(setuptools_version) < Ve



# setup.cfg
setup_cfg_content = """
[develop]
{script_option}=$base/lib/{package_name}

[install]
{install_scripts_option}=$base/lib/{package_name}
""".format(
    package_name=package_name,
    script_option='script-dir' if use_dash_separated_options
    install_scripts_option='install-scripts' if use_dash_sepa
)
```

```
# setup.cfg
with open("setup.cfg", "w") as f:
    f.write(setup_cfg_content)

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']),
        #Including the description files in share folder
        ('share/' + package_name + '/urdf'+'/mycobot_pro_630
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='todo', # we can also edit mantainer name and
    maintainer_email='todo@todo.todo',
    description='TODO: Package description', # as well as pac
    license='TODO: License declaration', # and license
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```

5. Building the package

We use *colcon* to build our package inside the workspace directory *~/colcon_ws/*

Note:

- colcon supports the option `--symlink-install`.
  This allows the installed files to be changed by changing the files in the

`source` space (e.g. Python files or other non-compiled resources) for faster iteration.

- We can either build all package or specifically ask for which package to be built using the flag `--packages-select <package_name>`

```
#Naviage to workspace directory
cd ~/colcon_ws/
# Build the package with symlink install
colcon build --symlink-install --packages-select mycobot630p
```

References

- ROS2 URDF Tutorial
- Collada (.dae) File Format
- MoveIt Robot Description