Step 3: Creating a ROS Packages for Robot Descriptions

Until now, we have placed URDF/xacro files along with our launch files into a single folder. This is not the recommended way to organize our `.xacro` files. There is a standard way to package a robot description into a ROS package. In this tutorial, we will learn how to create our own ROS package for our rrbot.

Standard Package Organization

This website contains information regarding a recommended structure of a ros2 robot. On that page there are a lot of extra details talking about ros2 control, which is a set of very powerful ros2 packages designed for real-time robot control. We won't be covering ros2 control in this course, but for students who are interested, I will share some ros2 control code for the ur3e robot arm during Lab 4. Anyways, here is a summary of the package structure that we will create for the `rrbot`. It is heavily inspired by the link above but with some slight tweaks.

```
├── <robot_name>_description/                           # Robot's
description files
│   ├── package.xml
│   ├── setup.py
│   ├── setup.cfg
│   ├── config/                                         # general YAML files
for a robot
│   │   └── <robot_name>_<someting_specific>.yaml
│   ├── launch/                                         # launch files
related to testing robots' description
│   │   └── view_<robot_name>.launch.py
│   ├── rviz/                                           # rviz display
configurations
│   │   └── <robot_name>_default.rviz
│   └── urdf/                                           # URDF file for the
robot
│       ├── materials.xacro                             # Common XACRO
definitions
│       ├── <robot_name>.urdf.xacro                     # Main URDF for a
robot - loads macro and other files
│       └── <robot_name>_macro.xacro
```

Creating the Description Package

Before we create the description package, we need to create a workspace for the package to live in.

New workspace setup

Create a new workspace folder with the following command.

```
cd ~/ece569-fall2025/Lab2
mkdir -p ws2/src
```

You should also edit your `~/.bashrc` file to point to this workspace, so that you can use the handy ros2 aliases you already created. For more details, see Lab1, step 3.

```
code ~/.bashrc
```

```
export ROS_WS=~/ece569-fall2025/Lab2/ws2/ # <--- change this line in your
~/.bashrc file
```

Make sure to open a new terminal for these changes to take effect.

Create the Description Package

Now, create your description package.

```
rosds
ros2 pkg create --build-type ament_python rrbot_description
```

Ensure that the package builds.

```
cb
```

Modifying setup.py

Now create the folder structure.

```
rosds
cd rrbot_description
mkdir -p config launch rviz urdf
```

Now, modify the `setup.py` file to include these folders you just created.

```
import os
from glob import glob
from setuptools import find_packages, setup

package_name = 'rrbot_description'
```

```
# ...
    data_files=[
        # ...
        (os.path.join('share', package_name, 'config'), glob('config/*')),
        (os.path.join('share', package_name, 'launch'), glob('launch/*')),
        (os.path.join('share', package_name, 'rviz'), glob('rviz/*')),
        (os.path.join('share', package_name, 'urdf'), glob('urdf/*')),
    ],
# ...
```

Moving other files

There are five other files in the Step3 starter code (besides this README.md file). Put them into the correct folders (use the standard package organization above to help you). There are some slight differences between this code and Step2, so be sure to use the code provided in Step3! You should also pay attention to how the new `rrbot_macro.xacro` code uses the `rrbot_dimension.yaml` file to define the xacro properties.

When you are done, build the package and open up RViz.

```
cb
ros2 launch rrbot_description view_rrbot.launch.py
```

RViz will open but the rrbot won't be visible just yet. Be sure to change the fixed frame to world, add TF, and add RobotModel just like you're familiar with. When you have the RViz settings how you like them, do the following:

1. Press `File -> Save Config As`
2. Navigate to the `ws2/src/rrbot_description/rviz/` folder (not the install folder!)
3. Save the config as `rrbot_default.rviz`.

When you are done, you can close the RViz windows (kill the running processes with `Ctrl+C` if necessary) and then run

```
cb
```

to rebuild the packages in your ros2 workspace. Now, when you run

```
ros2 launch rrbot_description view_rrbot.launch.py
```

the `rrbot_default.rviz` file you just created will correctly be opened. (You might have to run `rosclean` before `cb`.)

Task 1

Create a package called `table_description` in the same manner as you just did for the rrbot inside your `ws2/src` workspace folder. (If you want to use a `.yaml` file to store the table dimensions, you can but it is not required.) Create a launch file and run it to view the table in RViz. The purpose of this exercise is really to learn how to create a proper description package.

- You may wish to remove the `joint_state_publisher_gui` from the launch file since all of your table joints are fixed. Take a screenshot of the entire RViz window for your lab report.

Free Response Questions

There are additional free-response questions for this problem. See the Overleaf template.

Push to GitHub

Push your work from the step to GitHub.

Next Step

Proceed to Step 4