Introduction to rclpy and Custom Packages

Garrett Wells

University of Idaho

January 25, 2023

1 Review

2 Lab
- Workspaces, Packages, Nodes
- Demo: Workspace and Package Setup
- Pynput
- Synchronous and Asynchronous Actions

Summary of Lecture 1

- iRobot Create3 hardware
- ROS2 Interfaces
- Coding Example

Key Information From Lecture 1

- ROS2 uses interfaces(topics, services, actions) to interact with robots
- Interfaces can be explored with ros2 CLI
- ROS2 Client Libraries ➟ custom packages

Key Information

What are we doing in this lecture?

1. How to setup workspace and make your own ROS2 packages
2. How to use synchronous vs. asynchronous actions
3. How to use Pynput to get keyboard input
4. How to use callbacks

Workspaces and Packages

Definition

A **workspace** is a directory containing ROS 2 packages.

Definition

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.

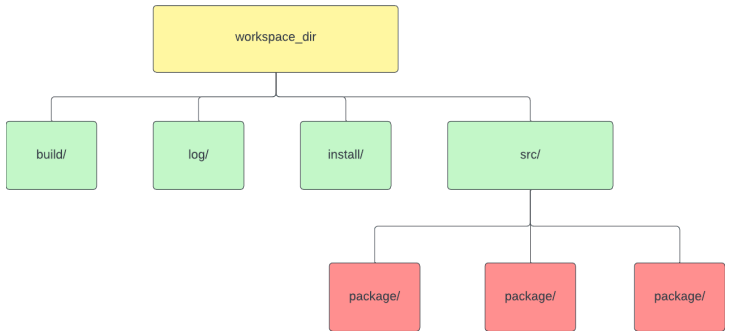

Figure: Typical workspace directory tree

▸ ROS2: Creating a Workspace

▸ ROS2: Creating a Package

Demo

How to create and build a new workspace from scratch.


```
1 source /opt/ros/humble/setup.bash
2 mkdir -p ~/ros2_ws/src
3 cd ~/ros2_ws/src
4 clone package dependencies ▸ irobot create msgs
5 rosdep install -i --from-path
  src --rosdistro humble -y
6 colcon build
```

Demo

How to create a new package from scratch.

1. source your ROS2 installation (step 1 of previous)
2. `cd ~/ros2_ws/src`
3. `ros2 pkg create --build-type ament_python <package_name>`
4. `cd ../ && colcon build`

Demo

How to configure your package.

1. Edit `package.xml` to add dependencies
   - `<exec_depend>rclpy</exec_depend>`
     `<exec_depend>irobot_create_msgs</exec_depend>`
2. Create and edit your source code.
3. **optional** Edit `setup.py` to add entry points to `console_scripts`
   - `'<name> = <pkg_name>.<script_name>:<function>'`
   - ex:
     `'linedriver = line_driver.main:main'`
4. switch back to build terminal session and `colcon build`

Nodes

- rclpy
- rclpy examples

Generic Usage...

```
from rclpy.node import Node
class <name>(Node):
    ...

def main():
    rclpy.init()
    node = ...
    rclpy.spin(node)
    rclpy.shutdown()
```

Adding Keyboard Control

Pynput Keyboard Control

```
from pynput import keyboard

def on_release(key):
    print('{0} released'.format(
        key))
    if key == keyboard.Key.esc:
        # Stop listener
        return False

listener = keyboard.Listener(
    on_press=None,
    on_release=on_release)

listener.start()
```

Synchronous Actions

Form

```
<action_client>.send_goal(<goal_obj>)
```

- ✔ requires the least code
- ✗ blocks, meaning that while the action is executing no messages may be received, no other jobs can be processed

Asynchronous Actions

Form

```
<action_client>.send_goal_async(<goal_obj>)
```

- ✔ provides ability to receive information from action server(robot or other node), process that information, and cancel action if needed
- ✗ callbacks are harder to trace/understand

How does this work?

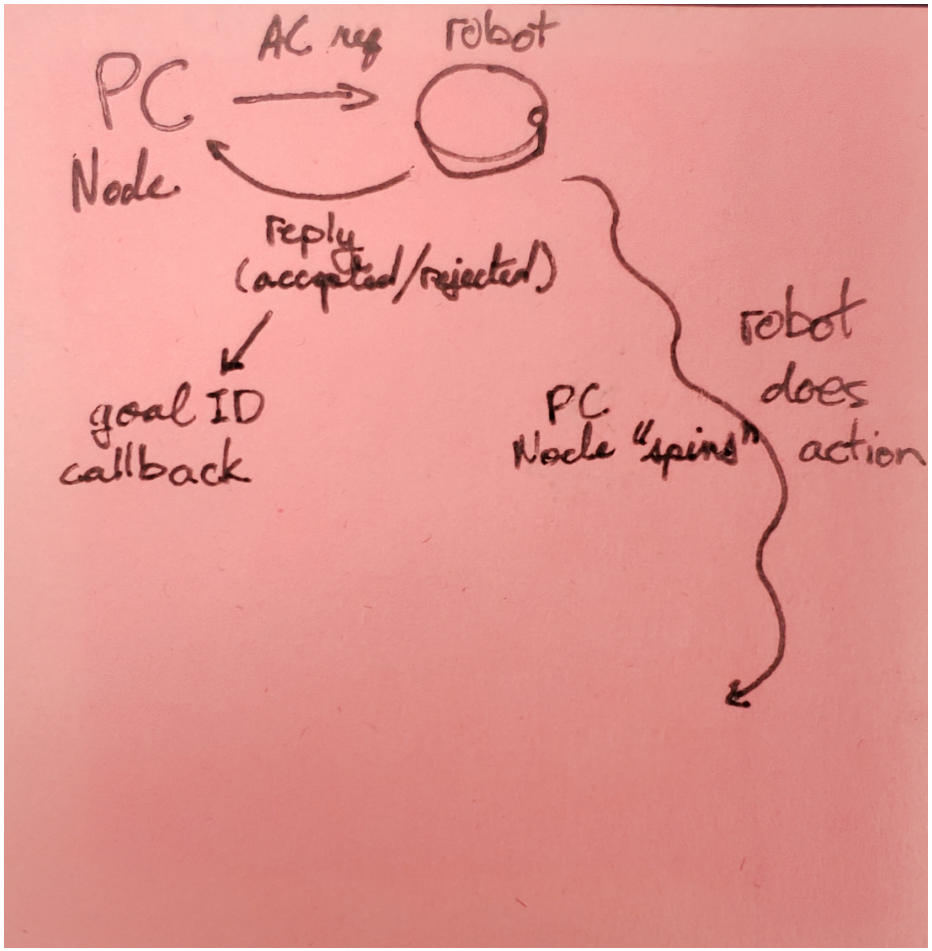

Figure: Artistic depiction of a robot's journey