ROS

“Robot Operating System”

Why do we use ROS?

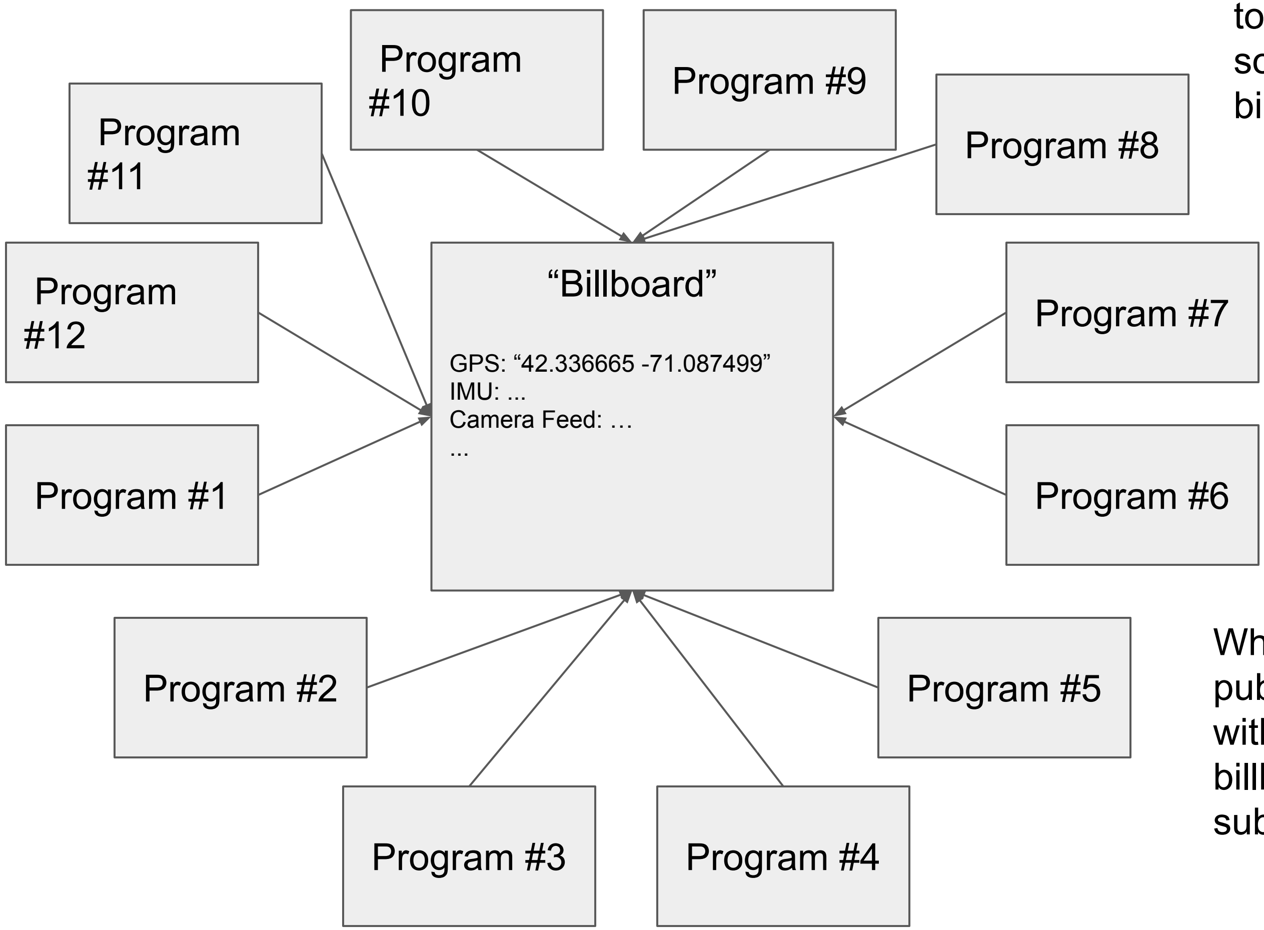
- The Rover is a complex system with lots of different parts/subsystems
  - Drive system, arm, autonomous, LD, camera streaming, etc…
- We want our codebase to be modular - the chunks of code that control each subsystem should be independent
  - Better for development and testing
- However, we need these chunks of code to be able to talk to each other, exchange data, etc
  - For example - the autonomous subsystem needs to talk to the drive and vice versa
- How do we do this?

What is ROS?

- Organizes a “workspace” of code into different packages
- Package have *nodes*, which can publish or subscribe to different messages
- This allows different chunks of code to talk to send and receive data without knowing exactly where they are sending/receiving from

Note: When we talk about ROS, we are talking about ROS 2. There are a few different versions of ROS 2 (we use Humble). There is also a version of ROS called ROS 1, which means there’s a lot of documentation out there. Here’s the right documentation link: https://docs.ros.org/en/humble/index.html

ROS Visualization:



Some programs **Publish** to the billboard, and some **Subscribe** to the billboard

When a publisher publishes to the billboard with a specific **Topic**, the billboard will notify each subscriber of that topic

2. Getting Started

A motor-themed demo project

Project Details

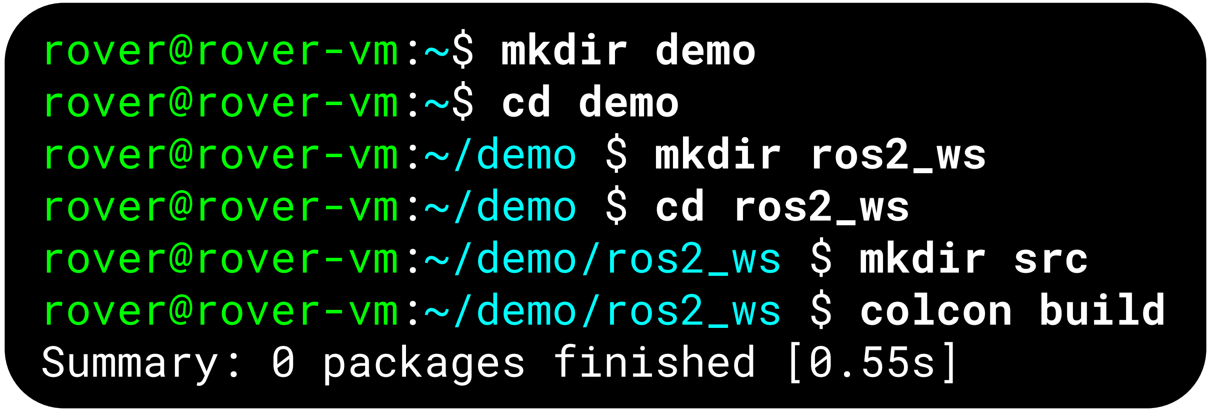
This project will control a fake motor represented by a “Motor” class. This motor can be controlled by sending it a degree to move to. The current position and the velocity can be read from the motor.

This project will have:

- A publisher node which sends degree commands
- A subscriber node which subscribes to this degree command and updates the motor position.

1. Create a Workspace

Workspace = directory that contains all of your packages. By convention this directory is named `ros2_ws`. Packages with your source code are put inside the `src` folder.



```
rover@rover-vm:~$ mkdir demo
rover@rover-vm:~$ cd demo
rover@rover-vm:~/demo $ mkdir ros2_ws
rover@rover-vm:~/demo $ cd ros2_ws
rover@rover-vm:~/demo/ros2_ws $ mkdir src
rover@rover-vm:~/demo/ros2_ws $ colcon build
Summary: 0 packages finished [0.55s]
```

2. Create a Package

- All ROS2 source code is placed into packages.
- There are two build types: “CMake” (for C++) and “Python”. Our demo will use Python.

```
rover@rover-vm:~/demo/ros2_ws $ cd src
rover@rover-vm:~/demo/ros2_ws/src $ ros2 pkg create --build-type
ament_python demo_package
going to create a new package
package name: demo_package
destination directory: /home/rover/demo/ros2_ws/src
package format: 3
[...]
creating ./hello_world/test/test_flake8.py
creating ./hello_world/test/test_pep257.py
rover@rover-vm:~/demo/ros2_ws/src $
```

3. Create a Publisher

- Because we are using Python, we will need to create a Python package inside a ROS package.

Directory structure: ros2_ws/src/demo_package/demo_package

- Each python package will have an __init__.py file in it (this is already generated by ROS2 when package is created with build type ‘ament_python’)
- Open your project in VSCode:

```
rover@rover-vm:~/demo/ros2_ws/$ code ~/demo/
```

- In the inner ‘demo_package’ folder, create a new file called “motor_degree.py”
- Copy the contents of the motor_degree.py file from this drive link here

Follow along for a live coding demo (or look at the rest of the slides)!

Motor Degree Publisher source code:

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32

class MotorDegreePublisher(Node):
   TIMER_PERIOD = 0.1
   DEGREE_INCR = 0.1

   def __init__(self):
       super().__init__('motor_degree_publisher')
       self.degree_publisher = self.create_publisher(Float32,
'/motor_degree_cmd', 1)

       self.timer = self.create_timer(self.TIMER_PERIOD,
self.update_degree_cmd)

       self.degree_cmd = Float32()

   def update_degree_cmd(self) -> None:
        """Is called every TIMER_PERIOD seconds to update deg cmd
value and publish cmd"""
       self.degree_cmd.data = (self.degree_cmd.data +
self.DEGREE_INCR) % 360
       self.degree_publisher.publish(self.degree_cmd)


def main(args=None):
   rclpy.init(args=args)
   motor_degree_publisher = MotorDegreePublisher()
   rclpy.spin(motor_degree_publisher)
```

What we did:

- Created a *publisher* called `degree_publisher`. This will publish a message of type “Float32” to the message name ‘/motor_degree_cmd’.
- Created a *timer* which will call the `update_degree_cmd` method every TIMER_PERIOD seconds.
- Created the `update_degree_cmd` method which updates the value of the degree_cmd and then *publishes* the message

3. Add an Entry Point

- You'll need to define an entry point for ROS to run your node
- You can do this in the setup.py file
- In a real project, you'd want to update the maintainer, email, etc. as well
- We're creating an entry point named `motor_degree`

```python
from setuptools import find_packages, setup

package_name = 'demo_package'

setup(
   name=package_name,
   version='0.0.0',
   packages=find_packages(exclude=['test']),
   data_files=[
       ('share/ament_index/resource_index/packages',
           ['resource/' + package_name]),
       ('share/' + package_name, ['package.xml']),
   ],
   install_requires=['setuptools'],
   zip_safe=True,
   maintainer='YOUR-NAME-HERE',
   maintainer_email='YOUR-EMAIL-HERE',
   description='TODO: Package description',
   license='TODO: License declaration',
   tests_require=['pytest'],
   entry_points={
       'console_scripts': [
           'motor_degree = demo_package.motor_degree:main',
       ],
   },
)
```

Running the Publisher:

You can use a command called ‘ros2 run’ to run ROS nodes. The command format is: “ros2 run <package name> <node name>”
Make sure you are in the ~/demo/ros2_ws directory!

```
rover@rover-vm:~/demo/ros2_ws/src $ cd ..
rover@rover-vm:~/demo/ros2_ws $ colcon build
output...
rover@rover-vm:~/demo/ros2_ws $ source install/setup.bash
rover@rover-vm:~/demo/ros2_ws $ ros2 run demo_package motor_degree
```

Listening for messages

Open up a second terminal - make sure you cd into the demo/ros2_ws directory

```
rover@rover-vm:~/demo/ros2_ws $ source install/setup.bash
rover@rover-vm:~/demo/ros2_ws $ ros2 node list
/motor_degree_publisher
rover@rover-vm:~/demo/ros2_ws $ ros2 topic list
/motor_degree_cmd
/parameter_events
/rosout
rover@rover-vm:~/demo/ros2_ws $ ros2 topic echo /motor_degree_cmd
data: 2.5999999046325684
---
data: 2.700000047683716
---
data: 2.799999952316284
---
data: 2.9000000953674316
---
data: 3.0
```

4. Create a Subscriber

- Create another file in the inner “demo_package” folder called “motor_controller.py”
- Copy the contents of the motor_controller.py file in the drive link here

Follow along for a live coding demo (or look at the rest of the slides)!

Subscriber Source Code:

```python
class MotorController (Node):
   def __init__ (self):
       # Step 1. Subscriber for degree command
       super().__init__('motor_controller')
       self.cmd_sub = self.create_subscription (
           Float32,
           '/motor_degree_cmd' ,
           self.handle_cmd,
           1
       )
       # Create an instance of the 'motor' to move
       self.motor = MockMotor()

   def handle_cmd(self, cmd: Float32) -> None:
       """Moves the 'motor' to the specified command and publishes state."""

       # log the info so we can see it!
       self.get_logger().info(f'Moving motor to {cmd.data} degrees')
       self.motor.move(cmd.data)

   def update_motor_state (self):
       """Updates the stored state of the 'motor' controlled by this class."""
       # this is for part 2!
       pass

def main(args=None):
   rclpy.init(args=args)
   motor_controller = MotorController()
   rclpy.spin(motor_controller)
```

Add an Entry Point

```python
from setuptools import find_packages, setup

package_name = 'demo_package'

setup(
   name=package_name,
   version='0.0.0',
   packages=find_packages(exclude=['test']),
   data_files=[
       ('share/ament_index/resource_index/packages',
           ['resource/' + package_name]),
       ('share/' + package_name, ['package.xml']),
   ],
   install_requires=['setuptools'],
   zip_safe=True,
   maintainer='dani',
   maintainer_email='YOUR-EMAIL-HERE',
   description='TODO: Package description',
   license='TODO: License declaration',
   tests_require=['pytest'],
   entry_points={
       'console_scripts': [
           'motor_degree = demo_package.motor_degree:main',
           'motor_controller = demo_package.motor_controller:main',
       ],
   },
)
```

Running the Subscriber:

1. Run ‘colcon build’ and ‘source install/setup.bash’ again
2. Run the publisher again (see the “Running the Publisher”, or follow along live)
3. Run the subscriber

```
rover@rover-vm:~/demo/ros2_ws $ ros2 run demo_package motor_controller
[INFO] [1725764063.975672456] [motor_controller]: Moving motor to
1.100000023841858 degrees
[INFO] [1725764064.069066894] [motor_controller]: Moving motor to
1.2000000476837158 degrees
[INFO] [1725764064.169588020] [motor_controller]: Moving motor to
1.2999999523162842 degrees
[INFO] [1725764064.269444034] [motor_controller]: Moving motor to
1.399999976158142 degrees
[INFO] [1725764064.370180683] [motor_controller]: Moving motor to 1.5
degrees
```

3. Custom Messages

Details

- What if we want to publish information about the state of our motor: its position, its torque, etc.
- We *could* create multiple publishers, but that can get messy, and it introduces a lot of overhead*.
- Solution: put all of the important info into one custom message type, so it can be handled by a single publisher or subscriber.

**The details aren't important, but a lot of extra bytes of metadata are added to a message when its published. If you have a ton of publishers sending out messages x times a second (with x being a large-ish number), these extra bytes can add up.*

1. Create another Package

- You will need to make a *cmake* package instead of a python package
- You will make a directory inside this package called ‘msg’, which is where the custom message definition will live.

```
rover@rover-vm:~/demo/ros2_ws/ $ cd src
rover@rover-vm:~/demo/ros2_ws/src $ ros2 pkg create --build-type
ament_cmake demo_msgs
rover@rover-vm:~/demo/ros2_ws/src $ cd demo_msgs
rover@rover-vm:~/demo/ros2_ws/src/demo_msgs $ mkdir msg
```

- Make a new file inside the ‘msg’ directory called “MotorState.msg”

2. Define the message

- The “MotorState.msg” file will have two fields: position and velocity.
- Each field will have the type ‘float32’
- Just copy-paste these lines into your file (it’s pretty simple!)

```
float32 angle # degree value representing current motor position
float32 torque # Nm value representing current torque on motor
```

3. Update CMakeLists.txt

- Copy-paste the following code into CMakeLists.txt before the last line - the line that says `ament_package()`

```
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/MotorState.msg"
)
```

4. Update package.xml

- Copy-paste the following code into the package.xml file, inside the <package> tags

```
<build_depend>rosidl_default_generators</build_depend>

<exec_depend>rosidl_default_runtime</exec_depend>

<member_of_group>rosidl_interface_packages</member_of_group>
```

5. Updating motor_controller.py

Add this import to the top of the file:

```
from demo_msgs.msg import MotorState
```

In the __init__() method of the MotorController class, create a new publisher and field to store the message value:

```
self.motor_pub = self.create_publisher(MotorState, '/motor_state', 1)

self.motor_state = MotorState()
```

Add code in `update_motor_state` to update the motor state:

```
    self.motor_state.angle = self.motor.read_pos()
    self.motor_state.torque = self.motor.read_torque()
```

Add these lines to the handle_cmd method:

```
self.update_motor_state()

self.motor_pub.publish(self.motor_state)
```

MotorController Class Source Code

```python
class MotorController(Node):
    def __init__(self):
        super().__init__('motor_controller')
        self.cmd_sub = self.create_subscription(
            Float32,
            '/motor_degree_cmd',
            self.handle_cmd,
            1
        )

        self.motor = MockMotor()

        self.motor_pub = self.create_publisher(MotorState, '/motor_state', 1)
        self.motor_state = MotorState()

    def handle_cmd(self, cmd: Float32) -> None:
        self.get_logger().info(f'Moving motor to {cmd.data} degrees')
        self.motor.move(cmd.data)
        self.update_motor_state()
        self.motor_pub.publish(self.motor_state)


    def update_motor_state(self):
        self.motor_state.angle = self.motor.read_pos()
        self.motor_state.torque = self.motor.read_torque()
```

6. Viewing the Custom Messages

1. Rebuild and re-source (run ‘colcon build’ and then ‘source install/setup.bash’)
2. Run the publisher and subscriber as before.
3. Run the following commands in a new terminal:

```
rover@rover-vm:~/demo/ros2_ws $ ros2 topic list
/motor_degree_cmd
/motor_state
/parameter_events
/rosout
rover@rover-vm:~/demo/ros2_ws $ ros2 topic echo /motor_state
angle: 12.899999618530273
torque: 0.0
---
angle: 13.0
torque: 0.0
```

4. Launch Files

What is a launch file?

- How do we run lots of nodes at once?
- The Rover involves dozens of nodes running simultaneously
- We need something that can launch a lot of nodes at once
  - Some nodes need to be configured differently in different circumstances (e.g. a CPU usage reporting node that runs on both the Jetson Orin and the Base Station)
  - We'd like to be able to leave comments saying which node does what
  - We want to be able to split up the configuration across multiple files, instead of one big file
- This is where ROS launch files come in
  - Define an .xml file with nodes you want to launch
  - Include other launch files

Making the launch file

1. Create a new folder inside the `demo_package` folder (the path should be `ros2_ws/src/demo_package/launch/`)
2. Make a new file inside this folder called `motor_launch.xml`
3. Add the following code to the launch file:

```
<launch>
   <node pkg="demo_package" exec="motor_degree" name="motor_degree" />
   <node pkg="demo_package" exec="motor_controller" name="motor_controller" />
</launch>
```

Updating setup.py and package.xml

1. Add the code in dashed boxes to the setup.py
2. Add the following code to the package.xml file inside the <package> tags

```
<exec_depend>ros2launch</exec_depend>
```

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

package_name = 'demo_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join("share", package_name, "launch"),
            glob("launch/*.xml"))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='dani',
    maintainer_email='your-email-here',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'motor_controller =
demo_package.motor_controller:main',
            'motor_degree = demo_package.motor_degree:main',
        ],
    },
)
```

Running a launch file

Note: make sure the publisher and subscriber aren't already running (if they are, type Ctrl+C in the terminal(s) they are running in to stop them)

```
rover@rover-vm:~/demo/ros2_ws $ colcon build
Starting >>> demo_msgs [1.241s]
Finished <<< demo_msgs [0.66s]
Starting >>> demo_package
Finished <<< demo_package [0.72s]

Summary: 2 packages finished
rover@rover-vm:~/demo/ros2_ws $ source install/setup.bash
rover@rover-vm:~/demo/ros2_ws $ ros2 launch demo_package motor_launch.xml
[INFO] [launch]: All log files can be found below
/home/dani/.ros/log/2024-09-15-23-35-32-626400-rover-vm-37266
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [motor_degree-1]: process started with pid [37267]
[INFO] [motor_controller-2]: process started with pid [37269]
[motor_controller-2] [INFO] [1726457733.011236114] [motor_controller]: Moving motor
to 0.10000000149011612 degrees
```

5. An Intro to Our Repo

https://gitlab.com/nuseds/rover