Principles of Robot Autonomy I

Fundamentals of ROS and vectorized computation in Python

Announcements

- Section signup Due Today @ 12PM
  - Contact us (email, Ed, after lecture, …) if
    1. You join the course late and missed this deadline
    2. Neither of the time slots work for you
  - Section works in groups of 3 - 4 students
- Section time assignment will be released by Today 5PM

Fundamentals of ROS

ROS2

Robot Operating System

Agenda

- Working with UNIX Terminal
- ROS2
  - Workspace & package
  - Executables
  - Communication -- publication & subscription
  - Working with third-party libraries
  - Dependencies Management with `rosdep`
- Vectorized Computation with `numpy`

A Quick Intro to UNIX Terminal

- Manipulating the filesystem in terminal
  - Filesystem commands
    - `cd` – change working directory
    - `ls` – list all files and sub-directories
    - `pwd` – get current working directory
    - `mkdir` – create directory
  - Absolute paths
    - E.g. `/home/aa274/Downloads/some_program.py`
  - Relative paths
    - E.g. `../Desktop/some_video.mp4`

A Quick Intro to UNIX Terminal

- All executables are just files with the right permission!
  - E.g. `ls -l /bin/ls` gives `-rwxr-xr-x` (note the x permisions)
- Running an executable file by specifying the path to the file
  - E.g. `/home/aa274/my_ws/awesome_program.py`
- To make a file executable
  - `chmod +x xxx.py`

A Quick Intro to UNIX Terminal

- Environment Variables
  - `export SOME_VAR=<some_value>`
  - `echo $SOME_VAR`
- The `source` command
  - `source some_script.bash`

ROS Workspace Structure

- Look at `~/tb_ws` in your local environment

ROS Workspace Structure

- `autonomy_ws/`
  - `src/`
    - `<repo1>/`
      - `<pkg1>`
      - `<pkg2>`
      - …
    - `<pkg3>/`
    - …
  - `install/`
  - `build/`
  - `log/`

Create a ROS Workspace

- `autonomy_ws/`
  - `src/` - create this directory
    - `<repo1>/`
      - `<pkg1>`
      - `<pkg2>`
      - …
    - `<pkg3>/`
    - …

Create a ROS Package

- `autonomy_ws/`
  - `src/`
    - `<repo1>/`
      - `<pkg1>`
      - `<pkg2>`
      - …
    - `<pkg3>/` - create your package here
    - …

```
ros2 pkg create --build-type ament_cmake <package name>
```

ROS Package Layout

- `your_package/`
  - `CMakeLists.txt` -- install scripts / link libraries
  - `package.xml` -- specify dependencies
  - ~~`include/`~~
  - ~~`src/`~~

ROS Package – Add a Python Script

- some_pkg/
  - CMakeLists.txt -- install scripts / link libraries
  - package.xml -- specify dependencies
  - scripts/
    - heartbeat.py

Registration of Executables in UNIX

- How are programs discovered in UNIX?
  - Executable Permission
    - `chmod +x <path to file>`
  - `PATH` environment variable
    - `echo $PATH`
    - `export PATH=<directory_of_executable>:$PATH`
  - The Shebang required on the top of any executable file
    - `#!/usr/bin/env python3`

Using ROS as a Python Library?

- Can I just run ROS python scripts normally (`./some_script.py`)?
    - Yes, I "sort of" did it along the way! Good for prototyping single component.
    - No, not a good practice in general. Especially if you want to integrate it to run with other scripts in the autonomy stack.

Registration of Executables in ROS

- `some_pkg/`
  - `CMakeLists.txt` -- install scripts / link libraries
  - `package.xml` -- specify dependencies
  - `scripts/`
    - `heartbeat.py`

Registration of Executables in ROS

- some_pkg/
  - CMakeLists.txt
  - package.xml
  - scripts/
    - heartbeat.py

```
install(PROGRAMS
    scripts/heartbeat.py
    DESTINATION lib/${PROJECT_NAME}
)
```

Build ROS Workspace

- `autonomy_ws/` -- current working directory needs to be here
  - `src/`
  - `install/`
  - `build/`
  - `log/`

```
1.   colcon build --symlink-install
2.   source install/setup.bash
```

Try to Run It through ROS

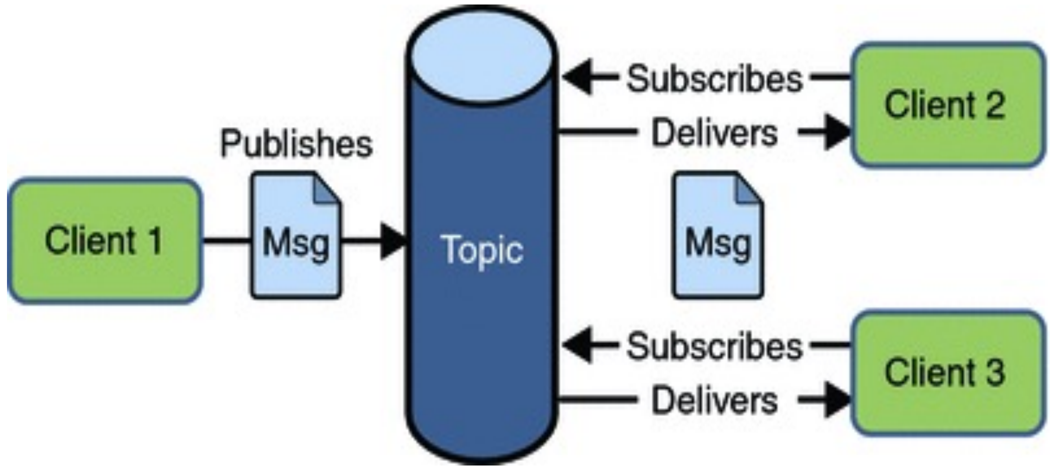
- `ros2 run <your_package> heartbeat.py`

Turn the Script into a ROS Node

```python
#!/usr/bin/env python3

import rclpy                                # ROS2 client library
from rclpy.node import Node                 # ROS2 node base class


class MinimalNode(Node):
    def __init__(self) -> None:
        # give it a default node name
        super().__init__("minimal_node")


if __name__ == "__main__":
    rclpy.init()                        # initialize ROS client library
    node = MinimalNode()                # create the node instance
    rclpy.spin(node)                    # call ROS2 default scheduler
    rclpy.shutdown()                    # clean up after node exits
```

ROS Communication

- Message Types
  - Data structure that holds some information about the robot
- Publication
  - Broadcast message to the ROS network
- Subscription
  - Listens to some broadcasted channel



ROS Communication - Messages

- ROS2 Common Interfaces
  - std_msgs
  - geometry_msgs
  - nav_msgs
  - sensor_msgs
  - …

ROS Communication - Messages

- ROS2 messages are data structures that are passed between nodes
  - Message types can be nested

```
float64 x
float64 y
float64 z
```

Vector3

```
Vector3 linear
Vector3 angular
```

Twist

```
Twist twist
float64[36] covariance
```

TwistWithCovariance

```python
from geometry_msgs.msg import TwistWithCovariance

msg = TwistWithCovariance()
msg.twist.angular.x = ...
```

ROS Communication - Messages

- You can create custom message types!
  - See here for some examples

ROS Communication - Publication

- Write a node that send out a “heartbeat” counter every second

```
# import the message type to use
from std_msgs.msg import Int64

# create publisher inside __init__ constructor
self.hb_pub = self.create_publisher(Int64, "/heartbeat", 10)

# publish message in a class method
msg = Int64()
msg.data = 10
self.hb_pub.publish(msg)
```

ROS Communication - Publication

- Recall from last lecture, use timer to trigger periodic events

```python
# create the timer and specify period in seconds
self.hb_timer = self.create_timer(1.0, self.hb_callback)

# create the callback function triggered by a timer
def hb_callback(self) -> None:
    # publish the heartbeat here
    ...
```

ROS Communication - Publication

- ROS2 CLI tools

```
# topic inspection
ros2 topic list
ros2 topic info <topic>
ros2 topic hz <topic>
ros2 topic echo <topic>
ros2 topic type <topic>
ros2 topic pub <topic> <msg type> <msg data>

# node inspection
ros2 node list
ros2 node info <node>
```

ROS Communication - Subscription

```python
# import the message type to use
from std_msgs.msg import Bool

# create subscription inside __init__ constructor
self.motor_sub = self.create_subscription(Bool, "/health/motor",
                                          self.health_callback, 10)

# create the callback function triggered by the subscription
def health_callback(self, msg: Bool) -> None:
    # stop your heartbeat if getting unhealthy sensors
    ...
```

Make the Heartbeat Stop

- Run the heartbeat node
  - `ros2 run <your_package> heartbeat.py`
- Publish an “unhealthy” sensor message
  - `ros2 topic pub /health/imu std_msgs/msg/Bool data:\ false -1`
- The heartbeat stops!

How does ROS Register Python Libraries?

- `asl_tb3_lib/`
  - `CMakeLists.txt`
  - `package.xml`
  - `asl_tb3_lib/`

```
ament_python_install_package(${PROJECT_NAME})
```

How to Import ROS Libraries?

- asl_tb3_lib/
  - CMakeLists.txt
  - package.xml
  - asl_tb3_lib/
    - __init__.py
    - control.py
    - tf_utils.py
    - ...

```
from asl_tb3_lib.control import BaseController
from asl_tb3_lib.tf_utils import yaw_to_quaternion
```

How are Python Libraries Discovered?

- `import numpy`
- `import numpiiiii?`
- `from ... import ...`

How are Python Libraries Discovered?

- `PYTHONPATH` environment variable
  - `echo $PYTHONPATH`
    - `import asl_tb3_lib` in a Python console
    - `unset PYTHONPATH` and run `import asl_tb3_lib` again
  - `export $PYTHONPATH=<...>:$PYTHONPATH`

How does ROS Package Handle Dependencies

- some_pkg/
  - CMakeLists.txt
  - package.xml
  - scripts/
    - heartbeat.py

```
<buildtool_depend>ament_cmake</buildtool_depend>
<buildtool_depend>ament_cmake_python</buildtool_depend>

<exec_depend>python3-termcolor</exec_depend>
<exec_depend>rclpy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>asl_tb3_lib</exec_depend>
<exec_depend>asl_tb3_msgs</exec_depend>
```

How does ROS Package Handle Dependencies

```
rosdep install --from-paths ~/autonomy_ws/src -i
```

Vectorized Computation with `numpy`

- Python loops are SLOW! (5x - 100x slower than C++)
- Avoid heavy computation with huge loops

Vectorized Computation with `numpy`

- Vectorized Mindset
  - Reduce loop to element-wise operations or numpy function calls
  - Array Slicing
  - Array Broadcasting

Vectorized Computation with `numpy`

- Vectorized Mindset
  - Reduce loop to element-wise operations or numpy function calls
    - Use built-in operators (`+, -, *, /, @, ...`)
    - Use Numpy APIs (e.g. `np.sum, np.prod, np.max, ...`)
  - Array Slicing
  - Array Broadcasting

Vectorized Computation with `numpy`

- Vectorized Mindset
  - Reduce loop to element-wise operations or numpy function calls
  - Array Slicing

```
x = np.zeros((10, 20))
x[3, 5]           # gives a scalar
x[3, :]           # gives a size-20 1d array
x[:, 3]           # gives a size-10 1d array
x[5:7, 10:15]     # gives a (2, 5) 2d array
x[None]           # gives a (1, 10, 20) 3d array
x[:, None, :]     # gives a (10, 1, 20) 3d array
x[..., None]      # gives a (10, 20, 1) 3d array
```

  - Array Broadcasting

Vectorized Computation with `numpy`

- Vectorized Mindset
  - Reduce loop to element-wise operations or numpy function calls
  - Array Slicing
  - Array Broadcasting

```
x = np.zeros((10, 20, 30))
y = np.zeros(30)
z = np.zeros((20, 1))
w = np.zeros((1, 20, 1))
# everything below result in (10, 20, 30) 3d arrays
x + y
x + z
x + w
```

Reminder

- Section starts tomorrow (Sept. 29)!
- Be sure to complete Skilling training form (See pinned Ed post)
- Check Edstem after 5 PM for section time assigment