CONTACT INFO

Fred Livingston, PhD

Email: fjliving@ncsu.edu

Mobile: 919.795.4710

Web: https://livingston.wordpress.ncsu.edu/

Bitbucket: https://bitbucket.org/livingston_ai/
SPRING 2023 WORKSHOP SERIES

- WS 001 – Introduction to Robot Programming using ROS2 [Feb 17th, 2023]
- WS 003 – Reactive Behaviors [TBD]
ROBOT PROGRAMMING USING ROS2

- Introduction to ROS2
- ROS Client Layer
- Simulation Tools
- robot program: architectures
- Task1: Basic control of ground vehicle
RESOURCES

- ROS/2 Tutorials
  - https://roboticsbackend.com/category/ros2
  - https://www.theconstructsim.com/robotigniteacademy_learnros/ros-courses-library/
PROGRAMMING ROBOTS

- Robots must be programmed to be useful
- We need Middlewares
- Robot programming middlewares provide drivers, libraries, and methodologies
- Few of them have survived the robot for which they were designed or have expanded from the laboratories where they were implemented
- The big difference is the ROS developers community around the world.
# ROS2 Design

## User Code

<table>
<thead>
<tr>
<th>User Code</th>
<th>User Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rclcpp</td>
<td>rclpy</td>
</tr>
<tr>
<td>rclc</td>
<td>Other APIs</td>
</tr>
</tbody>
</table>

## ROS Client Layer (RCL)

- rclcpp
- rclpy
- rclc
- Other APIs

| rcl (ROS2 C Implementation) |

## ROS Middleware Layer (RMW)

<table>
<thead>
<tr>
<th>rmw (DDS API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast DDS</td>
</tr>
<tr>
<td>Cyclone DDS</td>
</tr>
<tr>
<td>DDS</td>
</tr>
</tbody>
</table>

## OS Layer

- Windows
- Linux
- iOS
ROS DIMENSIONS

Workspace: the set of software installed on the robot or computer, the programs that the user develops, and tools to build

Community: vast community of developers who contribute with their own applications and utilities through public repositories, to which other developers can contribute

Computation Graph: a running ROS2 application
THE COMMUNITY

- Open Source and Licenses
- ROS2 organizes software development in federal model
- Packages and distributions
- Online resources
THE COMPUTATION GRAPH

• A robot's software looks like during its execution.
• A Computation Graph contains ROS2 nodes that communicate with each other so that the robot can carry out some tasks.
• The logic of the application is in the nodes, as the primary elements of execution in ROS2.
• Communication mechanisms:
  • Publication/Subscription: Asynchronous N:M
  • Services: Synchronous 1:1
  • Actions: Asynchronous 1:1
THE COMPUTATION GRAPH

- Execution model
  - Iterative
  - Event-Oriented

ROS

Robot Hardware
THE COMPUTATION GRAPH - EXAMPLE
• Approaches ROS2 software from a static point of view.
• Where the ROS2 software is installed, organized, and all the tools and processes that allow us to launch a computing graph.
• This includes the build system and node startup tools.
• Elements:
  • **Package**:  
    • It is the minimum functional set of software.
    • Contains executables, libraries, or message definitions with a common purpose.
  • **Workspace**:  
    • A directory that contains packages.
    • Activable to be available to use.
• **Underlay y overlay**
ROS2::FOXY INSTALATION

- VirtualBox (Option 1)
  - https://www.virtualbox.org/wiki/Downloads

- VMWare (Option 2 – Preferred)

- Ubuntu 20.04
  - https://releases.ubuntu.com/22.04/ubuntu-22.04.1-desktop-amd64.iso

- ROS2::Foxy (LTS)
VM — RECOMMENDED SETTINGS

Create Virtual Hard Disk

- File location: C:Users\Jiving\VirtualBox VMs\robotic.edu\robotic.edu.vdi
- File size: 2.00 TB

- Hard disk file type: VHD (Virtual Hard Disk)

Updates and other software

- What apps would you like to install to start with?
  - Normal installation
  - Minimal installation

- Other options:
  - Download updates while installing Ubuntu
  - This saves time after installation.
Make sure 3D Acceleration is Disabled
VM – RECOMMENDED SETTINGS

Install Guest Additions

© 2023 FRED LIVINGSTON. ALL RIGHTS RESERVED.
https://bitbucket.org/livingston_ai/
PYTHON 2 UNINSTALL (OPTIONAL)

- $sudo apt-get install python-is-python3
- $sudo apt-get autoremove --purge
$sudo
snap install code --classic
ROS2::FOXY INSTALLATION

https://bitbucket.org/livingston_ai/ros2_robot_programming/src/master/ros2_foxy_set up.ipynb?viewer=nbviewer
# ROS 2 Cheats Sheet

## Command Line Interface

All ROS 2 CLI tools start with the prefix `rostool` followed by a command, a verb and (possibly) positional/optional arguments.

For any tool, the documentation is accessible with,

```bash
$ rostool command --help
```

and similarly for verb documentation,

```bash
$ rostool command verb -h
```

Similarly, auto-completion is available for all commands/verbs and most positional/optional arguments. E.g.,

```bash
$ rostool command [tab][tab]
```

Some of the examples below rely on:

**ROS 2 demos package**

### action

Allows to manually send a goal and displays debugging information about actions.

**Verbs:**

- `info`  
  Output information about an action.
- `list`  
  Output a list of action names.
- `send goal`  
  Send an action goal.
- `show`  
  Output the action definition.

**Examples:**

```bash
$ rostool action info /fibonacci
$ rostool action list
$ rostool action send goal /fibonacci
  action.tutorials/action/Fibonacci "order: 5"
$ rostool action show action.tutorials/action/Fibonacci
```

### bag

Allows to record/play topics to/from a rosbag.

**Verbs:**

- `info`  
  Output information of a bag.
- `play`  
  Play a bag.
- `record`  
  Record a bag.

### interface

Various related verbs. Inte the following optional.

**Verbs:**

- `list`  
  List packages
- `package`  
  List packages
- `proto`  
  List proto
- `show`  
  List

**Examples:**

```bash
$ rostool interface
$ rostool interface
$ rostool interface
$ rostool interface
```

### launch

Allows to without to `cd` the

**Usage:**

```bash
$ rostool launch <
```

**Example:**

```bash
$ rostool launch di
```

### lifecycle

Various

**Verbs:**

- `get`  
  Get lifecycle state
- `list`  
  List lifecycle states
- `nodes`  
  List nodes
- `set`  
  Set lifecycle state
- `trig`  
  Trigger

**Examples:**

```bash
$ rostool doctor
$ rostool doctor --report
$ rostool doctor --report-fail
$ rostool doctor --include-warning
```

## Related Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Output a list of running containers and components.</td>
</tr>
<tr>
<td>load</td>
<td>Load a component into a container node.</td>
</tr>
<tr>
<td>standalone</td>
<td>Run a component into its own standalone container node.</td>
</tr>
<tr>
<td>types</td>
<td>Output a list of components registered in the ame index.</td>
</tr>
<tr>
<td>unload</td>
<td>Unload a component from a container node.</td>
</tr>
</tbody>
</table>
$ ros2 msg list
$ ros2 msg package std_msgs
$ ros2 msg packages
$ ros2 msg show geometry_msgs/msg/Pose

**multicast** Various multicast related verbs.
Verbs:
- **receive** Receive a single UDP multicast packet.
- **send** Send a single UDP multicast packet.

**node** Displays debugging information about nodes.
Verbs:
- **info** Output information about a node.
- **list** Output a list of available nodes.
Examples:
- $ ros2 node info /talker
- $ ros2 node list

**param** Allows to manipulate parameters.
Verbs:
- **delete** Delete parameter.
- **describe** Show descriptive information about declared parameters.
- **dump** Dump the parameters of a given node in yaml format, either in terminal or in a file.
- **get** Get parameter.
- **list** Output a list of available parameters.
- **set** Set parameter.
Examples:
- $ ros2 param delete /talker /use.sim_time
- $ ros2 param get /talker /use.sim_time
- $ ros2 param list
- $ ros2 param set /talker /use.sim_time false

**pkg** Create a ros2 package or output package(s)-related information.
Verbs:
- **create** Create a new ROS2 package.
- **is** Check if a package exists.
- **list** Output a list of available packages.
- **show** Show details of a package.
Examples:
- $ ros2 pkg create demo_nodes
- $ ros2 pkg list
- $ ros2 pkg prefix std_msgs
- $ ros2 pkg xml -t version

**run** Allows to run an executable in an arbitrary package without having to `cd` there first.
Usage:
- $ ros2 run <package> <executable>
Example:
- $ ros2 run demo_node.cpp talker

**security** Various security related verbs.
Verbs:
- **create_key** Create key.
- **create_permission** Create keystore.
- **generate_artifacts** Create permission.
- **list_keys** List keys.
- **create_keystore** Create keys and permission files from a list of identities and policy files.
- **generate_policy** Generate XML policy file from ROS graph data.
Examples (see `ros2 package`):
- $ ros2 security create_key demo_keys /talker
- $ ros2 security create_permission demo_keys /talker /polices/sample_policy.xml
- $ ros2 security generate_artifacts
- $ ros2 security create_keystore demo_keys

**service** Allows to manually call a service and displays debugging information about services.
Verbs:
- **call** Call a service.
- **find** Output a list of services of a given type.
- **list** Output a list of service names.
Examples:
- $ ros2 service call talker /use.sim_time /data: Hello RC
- $ ros2 service find talker
- $ ros2 service list talker
ROS CLIENT LAYER (RCL)

User Code

<table>
<thead>
<tr>
<th>ROS Client Layer (RCL)</th>
<th>User Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rclcpp</td>
<td>rclpy</td>
</tr>
<tr>
<td>rcl</td>
<td>Other APIs</td>
</tr>
<tr>
<td>rcl (ROS2 C Implementation)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROS Middleware Layer (RMW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmw (DDS API)</td>
</tr>
<tr>
<td>Fast DDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
</tr>
</tbody>
</table>
`$ ros2`

usage: ros2 [-h] Call ‘ros2 <command> -h’ for more detailed usage. ... ros2 is an extensible command-line tool for ROS 2.

...
$ ros2 pkg list
ackermann_msgs
action_msgs
action_tutorials_cpp
...

$ ros2 pkg executables demo_nodes_cpp
demo_nodes_cpp add_two ints_client
demo_nodes_cpp add_two ints_client async
demo_nodes_cpp add_two ints server
demo_nodes_cpp allocator tutorial
...
demo_nodes_cpp talker
...
RUNNING A ROS2 PROGRAM (PUBLISHER)

```
$ ros2 run demo_nodes_cpp talker

[INFO] [1643218362.316869744] [talker]: Publishing: 'Hello World: 1'
[INFO] [1643218363.316915225] [talker]: Publishing: 'Hello World: 2'
[INFO] [1643218364.316907053] [talker]: Publishing: 'Hello World: 3'
...
```

/talker

1 Hz

/chatter

std_msgs/msg/String
RUNNING A ROS2 PROGRAM

$ ros2 node list
/talker

$ ros2 topic list
/chatter
/parameter_events
/rosout
RUNNING A ROS2 PROGRAM

$ ros2 node info /talker

/talker
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /chatter: std_msgs/msg/String
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
  ...

/service /talker
  1 Hz

/service /chatter
  std_msgs/msg/String
RUNNING A ROS2 PROGRAM

$ ros2 topic info /chatter
Type: std_msgs/msg/String
Publisher count: 1
Subscription count: 0
INTERFACES

$ ros2 interface list

Messages:
  ackermann_msgs/msg/AckermannDrive
  ackermann_msgs/msg/AckermannDriveStamped
  ...
  visualization_msgs/msg/MenuEntry

Services:
  action_msgs/srv/CancelGoal
  ...
  visualization_msgs/srv/GetInteractiveMarkers

Actions:
  action.tutorials.interfaces/action/Fibonacci
  ...

$ ros2 interface show std_msgs/msg/String

... comments

string data
$ ros2 topic echo /chatter

data: 'Hello World: 1578'
---
data: 'Hello World: 1579'
...

/talker

1 Hz

/chatter

std_msgs/msg/String
RUNNING A SUBSCRIBER

$ ros2 run demo_nodes_py listener

[INFO] [1643220136.232617223] [listener]: I heard: [Hello World: 1670]
[INFO] [1643220137.197551366] [listener]: I heard: [Hello World: 1671]
[INFO] [1643220138.198640098] [listener]: I heard: [Hello World: 1672]

...
$ ros2 run rqt_graph rqt_graph
Native on Ubuntu 20.04

Install the following dependencies:

- ROS 2 Follow the instructions here to install ROS 2 Foxy.
- FITENTH Gym

```bash
git clone https://github.com/Fitenth/fitenth_gym
cd fitenth_gym && pip3 install -e .
```

Installing the simulation:

- Create a workspace: `cd $HOME && mkdir -p sim_ws/src`
- Clone the repo into the workspace:

```bash
cd $HOME/sim_ws/src
git clone https://github.com/Fitenth/fitenth_gym_ros
```

- Update correct parameter for path to map file: Go to `sim.yaml`
  `https://github.com/Fitenth/fitenth_gym_ros/blob/main/config/sim.yaml` in your cloned repo, change the `map_path` parameter to point to the correct location. It should be `'/your_home_dir/sim_ws/src/fitenth_gym_ros/maps/levine'`

- Install dependencies with rosdep:

```bash
source /opt/ros/foxy/setup.bash
cd ..
rosdep install -i --from-path src --rosdistro foxy -y
```

- Build the workspace: `colcon build`
$ git clone https://github.com/f1tenth/f1tenth_gym_ros
FIT TENTH GYM (CONFIGURATION)

Map_path: must contain full path name
Num_agent: 1 or 2
$ cd ..
$ source /opt/ros/foxy/setup.bash
$ rosdep install -i --from-path src --rosdistro foxy -y
FITENTH GYM (COMPILE SRC)

$ colcon build
F1TENTH GYM (RUN SIMULATOR)

$ ros2 launch f1tenth_gym_ros gym_bridge_launch.py
FITENTH GYM (RUN SIMULATOR)
TOPICS PUBLISHED BY THE SIMULATION

In single agent:

/scan: The ego agent's laser scan

/ego_racecar/odom: The ego agent's odometry

/map: The map of the environment

In two agents:

In addition to the topics available in the single agent scenario, these topics are also available:

/opp_scan: The opponent agent's laser scan

/ego_racecar/opp_odom: The opponent agent's odometry for the ego agent's planner

/opp_racecar/odom: The opponent agents' odometry

/opp_racecar/opp_odom: The ego agent's odometry for the opponent agent's planner
TOPICS SUBSCRIBED BY THE SIMULATION

In single agent:

/drive: The ego agent’s drive command via AckermannDriveStamped messages

/initalpose: This is the topic for resetting the ego’s pose via RViz’s 2D Pose Estimate tool. Do NOT publish directly to this topic unless you know what you’re doing.

In two agents:

In addition to all topics in the single agent scenario, these topics are also available:

/opp_drive: The opponent agent’s drive command via AckermannDriveStamped messages

/goal_pose: This is the topic for resetting the opponent agent’s pose via RViz’s 2D Goal Pose tool. Do NOT publish directly to this topic unless you know what you’re doing.
$ ros2 topic list -t

```
/robotdev@ubuntu:/sim_ws$ ros2 topic list -t

/cmd_vel [geometry_msgs/msg/Twist]
/clock [rosgraph_msgs/msg/Clock]
/drive [ackermann_msgs/msg/AckermannDriveStamped]
/ego_racecar/odom [nav_msgs/msg/Odometry]
/ego_robot_description [std_msgs/msg/String]
/goal_pose [geometry_msgs/msg/PoseWithCovarianceStamped]
/initial_pose [geometry_msgs/msg/PoseWithCovarianceStamped]
/joint_states [sensor_msgs/msg/JointState]
/map [nav_msgs/msg/OccupancyGrid]
/map_server/transition_event [lifecycle_msgs/msg/TransitionEvent]
/map_updates [nav_msgs/msg/OccupancyGridUpdate]
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/scan [sensor_msgs/msg/LaserScan]
/tf [tf2_msgs/msg/TFMessage]
/tf_static [tf2_msgs/msg/TFMessage]
```
GEOMETRY_MSGS/MSG/TWIST


geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```plaintext
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```plaintext
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

autogenerated on Wed, 02 Mar 2022 00:06:53
ackermann_msgs/AckermannDriveStamped Message

File: ackermann_msgs/AckermannDriveStamped.msg

Raw Message Definition

```
## Time stamped drive command for robots with Ackermann steering.
#
#
Header header
AckermannDrive drive
```

Compact Message Definition

```
std_msgs/Header header
ackermann_msgs/AckermannDrive drive
```
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
$ ros2 topic echo /cmd_vel
ROS2 Demos
https://github.com/ros2/demos
<?xml version="1.0"?>
<package format="2">
  <name>xero_nodes_py</name>
  <description>
    Python nodes which were previously in the ros2/examples repository but are now just used for demo purposes.
  </description>
  <maintainer email="aditya.pande@openrobotics.org">Aditya Pande</maintainer>
  <maintainer email="michael.jeronimo@openrobotics.org">Michael Jeronimo</maintainer>
  <license>Apache License 2.0</license>
  <author email="nathan@openrobotics.org">Nathan Fernandes</author>
  <author email="michael.jeronimo@openrobotics.org">Michael Jeronimo</author>
  <author>Mikel Arguiñano</author>
  <exec_depend>example_interases</exec_depend>
  <exec_depend>roslisp</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <build_type>ament_cmake</build_type>
</package>
CREATE A NEW PACKAGE

$ cd ~/sim_ws/src
$ ros2 pkg create my_robot_controller --build-type ament_python
SIMPLE PUBLISHER (TALKER.PY)

https://github.com/ros2/demos/blob/rolling/demo_nodes_py/demo_nodes_py/topics/talker.py

Line 25:
super().__init__('NAME_OF_PROCESS')

Line 27:
self.create_publisher(String, 'chatter', 10)

Line 29:
self.create_timer(1, self.time_call_back)
CREATE A PYTHON PROGRAM

$ cd my_robot_controller/my_robot_controller/
$ touch move_robot.py
$ gedit move_robot.py
$ cd my_robot_controller/my_robot_controller/
$ touch move_robot.py
$ gedit move_robot.py
def _init_(self):
    super().__init__(_name_="move_robot")
    self.pub = self.create_publisher(Twist, _cmd_vel_, 10)

    # move robot fwd
    msg = Twist()
    msg.linear.x = 0.5
    msg.linear.y = 0.0
    msg.linear.z = 0.0
    msg.angular.x = 0.0
    msg.angular.y = 0.0
    msg.angular.z = 0.0
    self.pub.publish(msg)

    timer_period = 10.0
    self.tmr = self.create_timer(timer_period, self.timer_callback)

def timer_callback(self):
    # stop robot
    msg = Twist()
    msg.linear.x = 0.0
    msg.linear.y = 0.0
    msg.linear.z = 0.0
    msg.angular.x = 0.0
    msg.angular.y = 0.0
    msg.angular.z = 0.0
    self.pub.publish(msg)
```python
from setuptools import setup

package_name = 'my_robot_controller'

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'])],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='robotdev',
    maintainer_email='Fred.Livingston@gmail.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['py.test'],
    entry_points={'console_scripts': [
        'move_robot = my_robot_controller.move_robot:main']},
)
```
BUILD AND EXECUTE ROBOT CONTROLLER

$ colcon build
$ source install/setup.bash
$ ros2 run my_robot_controller move_robot
END OF WORKSHOP

Fred Livingston (fjliving@ncsu.edu)