PROGRAMMING FOR AUTONOMOUS SYSTEMS

Fred Livingston, PhD
Work Shop 002
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 002 – Navigation [March 10th, 2023]
- WS 003 – Autonomous Navigation [TBD]
ROBOT NAVIGATION

- Review of ROS2
- F1TENTH Gym Setup
- Developing ROS Packages and Programs
- Mapping and Localization
ROS 2 Cheats Sheet

---

**Command Line Interface**

All ROS 2 CLI tools start with the prefix `ros2` followed by a command, a verb and (possibly) positional/optional arguments.

For any tool, the documentation is accessible with,

```
$ ros2 command --help
```

and similarly for verb documentation,

```
$ ros2 command verb -h
```

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

E.g.,

```
$ ros2 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:

```
$ ros2 action info /fibonacci
$ ros2 action list
$ ros2 action send_goal /fibonacci \ action.tutorials/action/Fibonacci "order: 5"
$ ros2 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.

---

**list** Output a list of running containers and components.

**load** Load a component into a container node.

**standalone** Run a component into its own standalone container node.

**types** Output a list of components registered in the current index.

**unload** Unload a component from a container node.

**Examples:**

```
$ ros2 component list
$ ros2 component load /ComponentManager \ composition:composition::Talker
$ ros2 component types
$ ros2 component unload /ComponentManager
```

**daemon** Various daemon related verbs.

Verbs:

- `start` Start the daemon if it isn’t running.
- `status` Output the status of the daemon.
- `stop` Stop the daemon if it is running.

**doctor** A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.

Alias: `wtf` (where’s the free).

Arguments:

- `--report/-r` Output report of all checks.
- `--report-fail/-rf` Output report of failed checks only.
- `--include-warning/-iw` Include warnings as failed checks.

Examples:

```
$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --report-fail
$ ros2 doctor --include-warning
```

**interface** Various related verbs. Inte the following optio srvs.

Verbs:

- `list` List
- `package` Ox
- `packages` Ox
- `te`
- `proto` Pr
- `for`
- `show` Ox

Examples:

```
$ ros2 interface
$ ros2 interface
$ ros2 interface
$ ros2 interface
$ ros2 interface
```

**launch** Allows to without to ‘cd’ the Usage:

```
$ ros2 launch <
```

Examples:

```
$ ros2 launch dir
```

**lifecycle** Various related verbs.

Verbs:

- `get` Get li
- `list` Outp
- `nodes` Outp
- `set` Trig

---

**msg** (deprecated)

Verbs:
$ 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.
- **executables** Create executables package.
- **list** List packages.
- **prefix** List prefix of a given std msg.
- **xml -t version** List xml 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** Generate artifacts.
- **list_keys** List keys.
- **create_keystore** Generate keys and permission files from a list of identities and policy files.
- **generate_key** Generate XML policy file from ROS graph data.
- **generate_policy** List keystore.

Examples (see `ros2 package`):
- $ ros2 security create_key demo.keys /talker
- $ ros2 security create_permission demo.keys /talker \ policies/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.

**test** Run a ROS2...

**topic** A tool for creating topics, including publishing and subscribing.

Verbs:
- **publish** Publish to topic.
- **subscribe** Subscribe to topic.
- **find** Find a topic.
- **hz** Find hz.
- **info** Output topic info.
- **list** Output topic list.
- **pab** Publish.
- **type** Output topic type.

Examples:
- $ ros2 topic publish topic
- $ ros2 topic subscribe
- $ ros2 topic find
- $ ros2 topic hz
- $ ros2 topic info
- $ ros2 topic list
- $ ros2 topic pab
- $ ros2 topic type
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/f1tenth/f1tenth_gym
cd f1tenth_gym && pip3 install -e .
```

Installing the simulation:

- Create a workspace:
  ```bash
cd $HOME && mkdir -p sim_ws/src
  
  cd $HOME/sim_ws/src
git clone https://github.com/f1tenth/f1tenth_gym_ros
  
  Update correct parameter for path to map file: Go to sim.yaml
  https://github.com/f1tenth/f1tenth_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/f1tenth_gym_ros/naps/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
FITENTH GYM (CONFIGURATION)

Map_path: must contain full path name
Num_agent: 1 or 2

```yaml
# copies or substantial portions of the Software.
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
bridge:
ros_parameters:
  # topics and namespaces
  egg_namespace: 'egg/rocecar'
  egg_scan_topic: 'scan'
  egg_odom_topic: 'odom'
  egg_opq_odom_topic: 'opp_odom'
  egg_drive_topic: 'drive'
  opp_namespace: 'opp/rocecar'
  opp_scan_topic: 'opp_scan'
  opp_odom_topic: 'odom'
  opp_opq_odom_topic: 'opp_odom'
  opp_drive_topic: 'opp_drive'
  # transform related
  scan_distance_to_base_link: 0.0
  # laser scan parameters
  scan_fov: 4.2
  scan_beams: 800
  # map parameters
  map_path: '/home/robotdev/sim_ws/src/fitenth_gym_ros/maps/levi/levi_map.lua'
  map_img_ext: 'png'
  # agent parameters
  num_agent: 1
  # egg starting pose on map
  sx: 0.0
  sy: 0.0
  stheta: 0.0
  # egg starting pose on map
  sx: 2.0
  sy: 0.5
  stheta: 9.0
  # opp starting pose on map
  sx: 2.0
  sy: 0.5
  stheta: 9.0
  # teleop
  kb_teleop: True
```
FIT TENTH GYM (INSTALL DEPENDENCIES)

$ cd ..
$ source /opt/ros/foxy/setup.bash
$ rosdep install -i --from-path src --rosdistro foxy -y
F1TENTH GYM (COMPILE SRC)

$ colcon build

```
robotdev@ubuntu:~/sim_ws$ ls
f1tenth_gym  src
robotdev@ubuntu:~/sim_ws$ colcon build
Starting >>> f110_gym
Starting >>> f1tenth_gym_ros
Finished <<< f110_gym [1.20s]
Finished <<< f1tenth_gym_ros [1.72s]
Summary: 2 packages finished [2.36s]
robotdev@ubuntu:~/sim_ws$
```
$ ros2 launch f1tenth_gym_ros gym_bridge_launch.py

robotdev@ubuntu:~/sim_ws$ ls
fitenth_gym  src
robotdev@ubuntu:~/sim_ws$ colcon build
Starting >>> f110_gym
Starting >>> f1tenth_gym_ros
Finished <<< f110_gym [1.20s]
Finished <<< f1tenth_gym_ros [1.72s]
Summary: 2 packages finished [2.36s]
robotdev@ubuntu:~/sim_ws$ ros2 launch f1tenth_gym_ros gym_bridge_launch.py
FITENTH GYM (RUN SIMULATOR)
$ ros2 topic list -t

```
robotdev@ubuntu:~/sim_ws$ ros2 topic list -t
/clicked_point [geometry_msgs/msg/PointStamped]
/clock [rosgallery_msgs/msg/Clock]
/cmd_vel [geometry_msgs/msg/Twist]
drive [ackermann_msgs/msg/AckermannDriveStamped]
ego_racecar/odom [nav_msgs/msg/Odometry]
ego_robot_description [std_msgs/msg/String]
goal_pose [geometry_msgs/msg/PoseStamped]
initialpose [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]
robotdev@ubuntu:~/sim_ws$  
```
TOPICS /TF

NEU (North, East, Up) Coordinates Systems
A useful function of the simulator is that you can instantly move the car without driving it to its new location. To do this, click the 2D Pose Estimate pose button at the top of the rViz window, and then click the desired location on the track to move the car there.
**geometry_msgs/Twist Message**

*File:* geometry_msgs/Twist.msg

**Raw Message Definition**

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

**Compact Message Definition**

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

* autogenerated on Wed, 02 Mar 2022 00:06:53*
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
$ ros2 topic echo /cmd_vel
**ackermann_msgs/AckermannDriveStamped Message**

File: `ackermann_msgs/AckermannDriveStamped.msg`

**Raw Message Definition**

```python
# Time stamped drive command for robots with Ackermann steering.
#
# $id$

Header  header
AckermannDrive  drive
```

**Compact Message Definition**

```python
std_msgs/Header header
ackermann_msgs/AckermannDrive drive
```
DEVELOPING PROGRAMS

ROS2 Demos
https://github.com/ros2/demos
<?xml version="1.0"?>
<package format="2">
  <name>xemo_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</email>
    <name>Aditya Pande</name>
  </maintainer>
  <maintainer>
    <email>mdadrow@openrobotics.org</email>
    <name>Michael Adrow</name>
  </maintainer>
  <maintainer>
    <email>mdelal@openrobotics.org</email>
    <name>Michael Delal</name>
  </maintainer>
  <license>Apache License 2.0/license>
  <author>
    <email>etei@openrobotics.org</email>
    <name>Jette Henningsen</name>
  </author>
  <author>
    <email>mdelal@openrobotics.org</email>
    <name>Michael Delal</name>
  </author>
  <author>
    <name>Mikel Argudin</name>
  </author>
  <exec_depend>example_interfas</exec_depend>
  <exec_depend>test_depend</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <export>
    <package></package>
  </export>
</package>
CREATE A NEW PACKAGE

$ cd ~/sim_ws/src
$ ros2 pkg create my_robot_controller --build-type ament_python
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
MOVE_ROBOT.PY

$ cd my_robot_controller/my_robot_controller/
$ touch move_robot.py
$ gedit move_robot.py
import rclpy
from rclpy.executors import ExternalShutdownException
from rclpy.node import Node

from geometry_msgs.msg import Twist

class Controller(Node):

def __init__(self):
    super().__init__('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.tm = 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

description='TODO: Package description',
license='TODO: License declaration',
tests_require=['pytest'],
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
SLAM — SIMULTANEOUS LOCALIZATION & MAPPING

SLAM is a technique used to build up a map within an unknown environment or a known environment while at the same time keeping track of the current location.
WHAT IS SLAM

• The problem has 2 stages
  • Mapping
  • Localization

• The paradox:
  • In order to build a map, we must know our position
  • To determine our position, we need a map!

• SLAM is like the chicken-egg problem

• Solution is to alternate between the two steps.
SLALM — MULTIPLE PARTS

- Landmark extraction
- Data association
- State estimation
- State update
- Landmark update

There are many ways to solve each of the smaller parts
THE GOAL OF THE PROCESS

The SLAM process consists of number of steps.

- Use environment to update the position of the robot. Since the odometry of the robot is often erroneous we cannot rely directly on the odometry.
- We can use laser scans of the environment to correct the position of the robot.
- This is accomplished by extracting features from the environment and re observing when the robot moves around.
An EKF (Extended Kalman Filter) is the heart of the SLAM process.

- It is responsible for updating where the robot thinks it is based on the Landmarks (features).

- The EKF keeps track of an estimate of the uncertainty in the robot's position and also the uncertainty in these landmarks it has seen in the environment.
SLAM OVERVIEW

Laser Scans

Odometry Change

EKF Odometry update

Landmark Extraction

Data Association

EKF Odometry update

EKF Re-observation

EKF New Observations

Robot

Landmarks

Laser scan
OVERVIEW

- Laser Scans
- Odometry Change
- EKF New Observations
- EKF Re-observation
- EKF Odometry update
- Landmark Extraction
- Data Association

Moved Robot (delta position)
OVERVIEW

Laser Scans

- Odometry Change
- EKF Odometry update
- EKF Re-observation
- EKF New Observations

- Landmark Extraction
- Data Association

Updated laser-scan @ new position
Laser Scans

Odometry Change

EKF Odometry update

Landmark Extraction

Data Association

EKF Re-observation

EKF New Observations

Pose based on laser data
Pose based on using odometry, note that odometry uses velocity to compute delta position and is less accurate than laser
OVERVIEW

Laser Scans

EKF to estimate robot position using odometry and laser

EKF Odometry update

EKF Re-observation

EKF New Observations

Landmark Extraction

Data Association

Odometry Change
Laser and Odometry Data

• Laser data is the reading obtained from the scan.
• The goal of the odometry data is to provide an approximate position of the robot.
• The difficult part about the odometry data and the laser data is to get the timing right.
REPRESENTATION

- Grid maps or scans

  [Lu & Milios, 97; Gutmann, 98; Thrun 98; Burgard, 99; Konolige & Gutmann, 00; Thrun, 00; Arras, 99; Haehnel, 01;...]

- Landmark-based

  [Leonard et al., 98; Castelanos et al., 99; Dissanayake et al., 2001; Montemerlo et al., 2002;...]
Landmarks are features which can easily be re-observed and distinguished from the environment. These are used by the robot to find out where it is (to localize itself).
KEY POINTS ABOUT SUITABLE LANDMARKS

- Landmarks should be easily re-observable.
- Individual landmarks should be distinguishable from each other.
- Landmarks should be plentiful in the environment.
- Landmarks should be stationary.
AUTONOMOUS NAVIGATION
WORK IN-PROGRESS

WS 003 – Autonomous Navigation [TBD]
It has tools to:

• **Load, serve, and store maps** (Map Server)
• **Localize the robot on the map** (AMCL)
• **Plan a path from A to B around obstacles** (Nav2 Planner)
• **Control the robot as it follows the path** (Nav2 Controller)
• **Smooth path plans to be more continuous and feasible** (Nav2 Smoother)
• **Convert sensor data into a costmap representation of the world** (Nav2 Costmap 2D)
• **Build complicated robot behaviors using behavior trees** (Nav2 Behavior Trees and BT Navigator)
• **Compute recovery behaviors in case of failure** (Nav2 Recoveries)
• **Follow sequential waypoints** (Nav2 Waypoint Follower)
• **Manage the lifecycle and watchdog for the servers** (Nav2 Lifecycle Manager)
• **Plugins to enable your own custom algorithms and behaviors** (Nav2 Core)
• **Monitor raw sensor data for imminent collision or dangerous situation** (Collision Monitor)
• **Python3 API to interact with Nav2 in a pythonic manner** (Simple Commander)
• **A smoother on output velocities to guarantee dynamic feasibility of commands** (Velocity Smoother)
INSTALLATION

SLAM Toolbox
$sudo apt install ros-foxy-slam-toolbox

ROS NAV2
$sudo apt install ros-foxy-navigation2
$sudo apt install ros-foxy-nav2-bringup
$sudo apt install ros-foxy-twist-mux
CONFIGURING SLAM TOOLBOX

/opt/ros/foxy/share/slam_toolbox
CONFIGURING SLAM TOOLBOX [MAPPING]

$ sudo nano /opt/ros/foxy/share/slam_toolbox/config/mapper_params_online_async.yaml
RUNING SLAM TOOLBOX [MAPPING]

$ cd ~/sim_ws

$ source install/setup.bash

$ ros2 launch slam_toolbox online_async_launch.py
SLAM TOOLBOX PLUGIN
SAVE MAP
CONFIGURING SLAM TOOLBOX [LOCALIZATION]

$ sudo nano /opt/ros/foxy/share/slam_toolbox/config/mapper_params_online_async.yaml
RUNING NAVIGATION

$ cd ~/sim_ws

$ source install/setup.bash

$ ros2 launch nav2_bringup navigation_launch.py
END OF WORKSHOP

Fred Livingston (fjliving@ncsu.edu)