Robotics and Intelligent Systems Lab

Week 3

Francesco Maurelli

Fall 2022

Recap

Recap

- Introduction to ROS & installation

Recap

- Introduction to ROS & installation
- ROS package system
  - Navigating: `rospackage`, `roscd`, `rosls`

Recap

- Introduction to ROS & installation
- ROS package system
  - Navigating: `rospackage`, `roscd`, `rosls`
- ROS packages & Catkin
  - creating a catkin workspace (`catkin build`)
  - workspace structure
  - creating and building a ROS package
  - customizing your package.xml

Recap

- `roscore`
  - launches master, parameter server, /rosout

Recap

- `roscore`
  - launches master, parameter server, /rosout
- `rosparam`
  - `get, set, load, dump, delete, list`

Recap

- `roscore`
  - launches master, parameter server, /rosout
- `rosparam`
  - `get, set, load, dump, delete, list`
- `rosnode`
  - `list, info, ping, kill, machine, cleanup`

Recap

- `roscore`
  - launches master, parameter server, /rosout
- `rosparam`
  - `get, set, load, dump, delete, list`
- `rosnode`
  - `list, info, ping, kill, machine, cleanup`
- `rosrun`
  - `rosrun <package_name> <node_name>`

Recap

- `roscore`
  - launches master, parameter server, /rosout
- `rosparam`
  - `get, set, load, dump, delete, list`
- `rosnode`
  - `list, info, ping, kill, machine, cleanup`
- `rosrun`
  - `rosrun <package_name> <node_name>`
- `rostopic`
  - `list, info, echo, type`

Recap

- `roscore`
  - launches master, parameter server, /rosout
- `rosparam`
  - `get, set, load, dump, delete, list`
- `rosnode`
  - `list, info, ping, kill, machine, cleanup`
- `rosrun`
  - `rosrun <package_name> <node_name>`
- `rostopic`
  - `list, info, echo, type`
- `rqt_graph`
  - graphical representation of nodes and topics

ROS Messages, Publishing and Plotting

ROS Messages I

- Messages define the structure of the data that is passed between nodes
- Standard datatypes such as `int64`, `int8`, `string`, `time`, `bool`, `float32`, `float64`
- There are several standard message types that come with commonly used packages
- Messages are defined in `.msg` files and are stored under the `<pkg>/msg` directory

ROS messages

- `std_msgs`: standard messages
- `geometry_msgs`: pose, point, Quat, etc.
- `nav_msgs`: Path, Odometry, Map, etc.
- `sensor_msgs`: Image, Imu, Laserscan, etc.

geometry_msgs/Twist

```
Vector3 linear
Vector3 angular
```

Vector3

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

ROS Messages II

- `rostopic type [/topic]` can be used to return the message type of any topic being published

```
▶ rostopic type /turtle1/cmd_vel
geometry_msgs/Twist
```

- `rosmsg info [/topic]` shows the message description

```
▶ rosmsg info geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

- combining both

```
$ rostopic type /turtle1/cmd_vel | rosmsg show
```

- try that with out topics & messages (geometry_msgs/Pose, sensor_msgs/Image)

ROS Messages III

- `rosmsg package <package_name>` can be used to return the messages within a certain package

```
$ rosmsg package turtlesim
turtlesim/Color
turtlesim/Pose
```

Publishing

- One can publish data to a topic from the command-line

```
$ rostopic pub [topic] [msg_type] [args]
```

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0,
0.0]' '[0.0, 0.0, 1.8]'
```

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '{linear:
{x:2.0,y:0.0,z:0.0}, angular:  {x:0.0,y:0.0,z:1.8}}'
```

Publishing

- One can publish data to a topic from the command-line

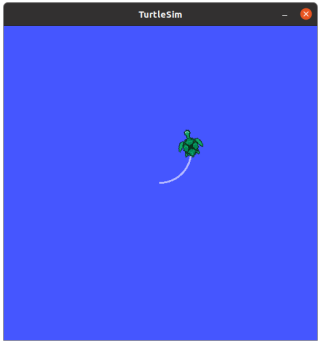
```
$ rostopic pub [topic] [msg_type] [args]
```

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0,
0.0]' '[0.0, 0.0, 1.8]'
```

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '{linear:
{x:2.0,y:0.0,z:0.0}, angular:  {x:0.0,y:0.0,z:1.8}}'
```

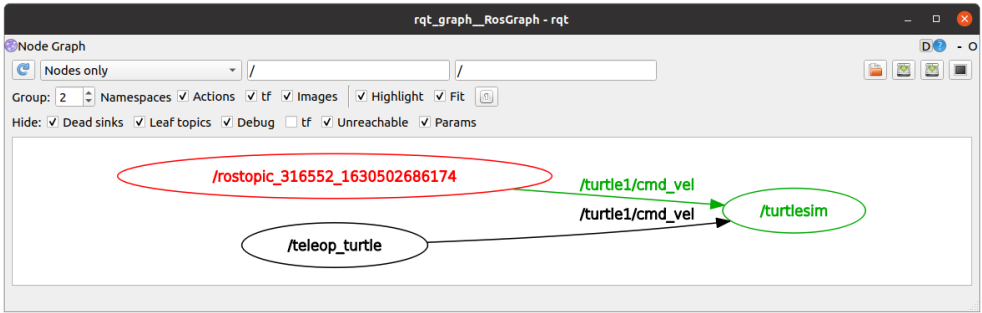
- Now try

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '{linear:
{x:2.0,y:0.0,z:0.0}, angular:  {x:0.0,y:0.0,z:-1.8}}'
```



Publishing

- Inspecting the `rqt_graph`



Node Rate

- `rostopic hz` is used to get the rate at which data is published

```
$ rostopic hz [topic]
```

```
$ rostopic hz /turtle1/pose
```

Node Rate

- `rostopic hz` is used to get the rate at which data is published

```
$ rostopic hz [topic]
```

```
$ rostopic hz /turtle1/pose
```

- Now try

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

Plotting

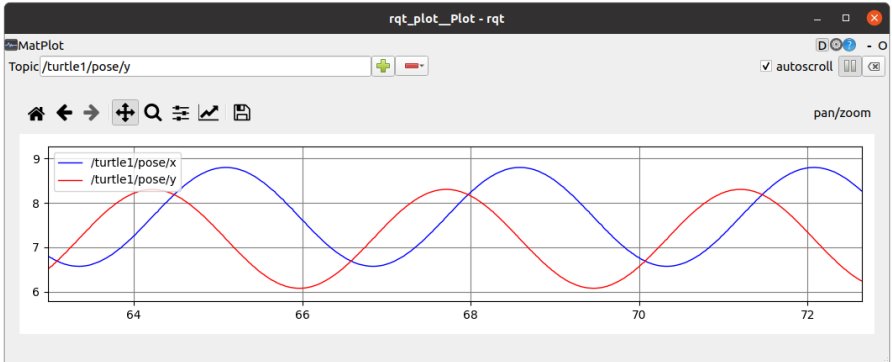
- `rqt_plot` can be used to display a scrolling time plot of the data published on topics

```
$ rosrun rqt_plot rqt_plot
```

Plotting

- `rqt_plot` can be used to display a scrolling time plot of the data published on topics

```
$ rosrun rqt_plot rqt_plot
```



Exercise: Get to know ROS

Exercise I

1. Setup the husky simulation
   - clone https://github.com/husky/husky into your `catkin_ws/src` and build it
   - Set an environmental variable HUSKY_GAZEBO_DESCRIPTION:

```
$ export HUSKY_GAZEBO_DESCRIPTION=$(rospack find husky_gazebo)/urdf/description.gazebo.xacro
```

   - install the lms1xx dependency

```
$ sudo apt-get install ros-noetic-lms1xx
```

Exercise II

2. Launch the simulation and inspect the created nodes and topics

```
$ roslaunch husky_gazebo husky_empty_world.launch
```

- `rosnode list`
- `rostopic list`
- `rostopic echo [topic]`
- `rostopic hz [topic]`
- `rostopic type [topic]`
- `rqt_graph`
- `ros_msg show`

Exercise III

3. Command a desired velocity to the robot from the terminal and plot the topic using `rqt_plot`

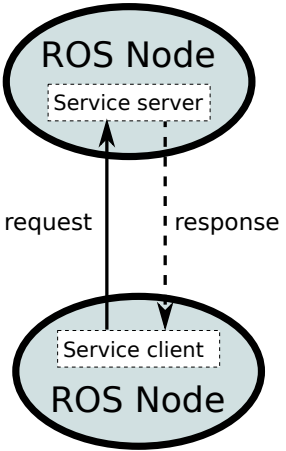
```
$ rostopic pub [topic]
```

4. Use teleop_twist_keyboard to control the robot using the keyboard
   - find the package online, clone and build it from source

Services & Parameters

ROS Service

- A Service implements a request/response mechanism for inter-node communication.
- **Service**/**Client** model: 1-to-1 request-response
- **Server** node: provides the service
- **Client** node: requests a response
- Services are used for sending infrequent signals to a node or for asking a node to perform a remote calculation
- The service type is defined in a *.srv file



Using ROS Services I

- `rosservice list`: list active services
- `rosservice info`: print information about service
- `rosservice type`: print service type
- `rosservice args`: print service arguments
- `rosservice call`: call the service with the provided args
- `rosservice find`: find services by service type
- `rosservice uri`: print service ROSRPC uri

Using ROS Services II

- let's list the list of services available (make sure turtlesim is running)

Using ROS Services II

- let's list the list of services available (make sure turtlesim is running)

```
▶ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

Using ROS Services II

- let's list the list of services available (make sure turtlesim is running)

```
▶ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

- inspect the type of a service

```
$ rosservice type <service_name>
```

Using ROS Services II

- let's list the list of services available (make sure turtlesim is running)

```
▶ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

- inspect the type of a service

```
$ rosservice type <service_name>
```

```
▶ rosservice type /turtle1/set_pen
turtlesim/SetPen
```

Using ROS Services III

- let's inspect the type of a certain service

```
$ rossrv show <service_type>
```

Using ROS Services III

- let's inspect the type of a certain service

```
$ rossrv show <service_type>
```

```
▶ rosservice type /turtle1/set_pen | rossrv show
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

Using ROS Services III

- let's inspect the type of a certain service

```
$ rossrv show <service_type>
```

```
▶ rosservice type /turtle1/set_pen | rossrv show
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

- let's call a service using

```
$ rosservice call /turtle1/set_pen 255 0 0 1 0
```

Using ROS Services III

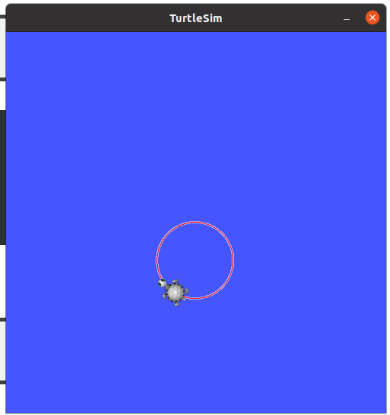
- let's inspect the type of a certain service

```
$ rossrv show <service_type>
```

```
▶ rosservice type /turtle1/set_pen | rossrv show
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

- let's call a service using

```
$ rosservice call /turtle1/set_pen 255 0 0 1 0
```



Using ROS Services III

- let's inspect the type of a certain service

```
$ rossrv show <service_type>
```

```
▶ rosservice type /turtle1/set_pen | rossrv show
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

- let's call a service using

```
$ rosservice call /turtle1/set_pen 255 0 0 1 0
```

- try the same for the service `/spawn`

ROS Parameters I

- The Parameter Server can store integers, floats, boolean, dictionaries, and lists
- `rosparam` can be used to manipulate and inspect parameters
- `rosparam` uses YAML markup language for syntax

ROS Parameters I

- The Parameter Server can store integers, floats, boolean, dictionaries, and lists
- `rosparam` can be used to manipulate and inspect parameters
- `rosparam` uses YAML markup language for syntax
- list of commands
  - `rosparam list`: list active parameters
  - `rosparam set`: set a parameter
  - `rosparam get`: get a parameter
  - `rosparam load`: load parameter from file
  - `rosparam dump`: dump parameter to file
  - `rosparam delete`: delete parameter

ROS Parameters II

- Let's look at what parameters are currently on the param server

ROS Parameters II

- Let's look at what parameters are currently on the param server

```
▶ rosparam list
/rosdistro
/roslaunch/uris/host_eurex_lap2_u__40999
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
```

ROS Parameters II

- Let's look at what parameters are currently on the param server

```
▶ rosparam list
/rosdistro
/roslaunch/uris/host_eurex_lap2_u__40999
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
```

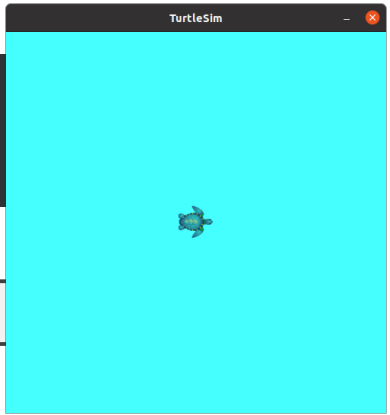
- Let's change the background of turtlesim using

```
$ rosparam set <param_name>
```

ROS Parameters II

- Let's look at what parameters are currently on the

```
▶ rosparam list
/rosdistro
/roslaunch/uris/host_eurex_lap2_u__40999
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
```



- Let's change the background of turtlesim using

```
$ rosparam set <param_name>
```

ROS Parameters III

- We can also get the contents of the entire Parameter Server using

```
$ rosparam get /
```

ROS Parameters III

- We can also get the contents of the entire Parameter Server using

```
$ rosparam get /
```

```
▶ rosparam get /
rosdistro: 'noetic

  '
roslaunch:
  uris:
    host_eurex_lap2_u__43401: http://EUREX-LAP2-U:43401/
rosversion: '1.15.11

  '
run_id: 08a75678-0ca5-11ec-8bfc-e1d513279c14
turtlesim:
  background_b: 255
  background_g: 255
  background_r: 69
```

ROS Parameters

- Parameters can also be stored into a file for later use

```
$ rosparam dump [file_name] [namespace]
$ rosparam load [file_name] [namespace]
```

ROS Parameters

- Parameters can also be stored into a file for later use

```
$ rosparam dump [file_name] [namespace]
$ rosparam load [file_name] [namespace]
```

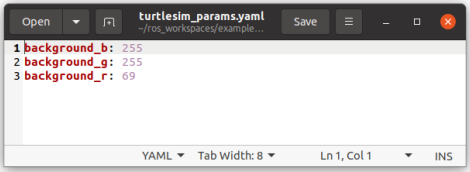
- Dump the values of turtlesim in a trurtlesim_params.yaml

ROS Parameters

- Parameters can also be stored into a file for later use

```
$ rosparam dump [file_name] [namespace]
$ rosparam load [file_name] [namespace]
```

- Dump the values of turtlesim in a trurtlesim_params.yaml

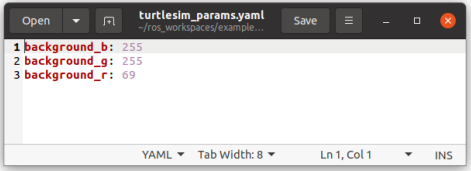


ROS Parameters

- Parameters can also be stored into a file for later use

```
$ rosparam dump [file_name] [namespace]
$ rosparam load [file_name] [namespace]
```

- Dump the values of turtlesim in a trurtlesim_params.yaml



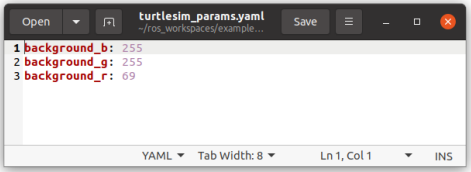
- Modify the generated file and load it back into the parameter server

ROS Parameters

- Parameters can also be stored into a file for later use

```
$ rosparam dump [file_name] [namespace]
$ rosparam load [file_name] [namespace]
```

- Dump the values of turtlesim in a trurtlesim_params.yaml



- Modify the generated file and load it back into the parameter server

```
$ rosparam load turtlesim_params.yaml /turtlesim
```

Review

- `rostopic`
  - `list`, `info`, `echo`, `type`, `pub`, `hz`

Review

- `rostopic`
  - `list`, `info`, `echo`, `type`, `pub`, `hz`
- `rosmsg`
  - `show(info)`, `list`, `package`

Review

- `rostopic`
  - `list, info, echo, type, pub, hz`
- `rosmsg`
  - `show(info), list, package`
- `rqt_graph`
  - graphical representation of nodes and topics

Review

- `rostopic`
  - `list, info, echo, type, pub, hz`
- `rosmsg`
  - `show(info), list, package`
- `rqt_graph`
  - graphical representation of nodes and topics
- `rqt_plot`
  - continuous stream plotting of data published on topics

Review

- `rostopic`
  - `list, info, echo, type, pub, hz`
- `rosmsg`
  - `show(info), list, package`
- `rqt_graph`
  - graphical representation of nodes and topics
- `rqt_plot`
  - continuous stream plotting of data published on topics
- `rossrv`
  - `list, show(info), type, call`

Review

- `rostopic`
  - `list, info, echo, type, pub, hz`
- `rosmsg`
  - `show(info), list, package`
- `rqt_graph`
  - graphical representation of nodes and topics
- `rqt_plot`
  - continuous stream plotting of data published on topics
- `rossrv`
  - `list, show(info), type, call`
- `rosparam`
  - `set, get, dump, load`