ROS Parameters

ECE 495/595 Lecture Slides

Winter 2017

Instructor: Micho Radovnikovich

Summary and Quick Links

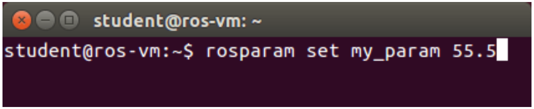
These slides contain the following concepts:

ROS Parameters

- The ROS parameter server allows the user to set values that are available for nodes to use at run-time.
- Parameters differ from topics in that they are static data that any node can access at any time, rather than being a node to node transmission of data.
- Parameter values can be set in a terminal, in a launch file, or in a YAML file.

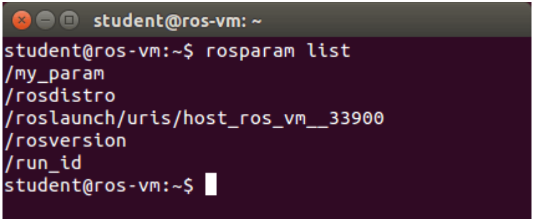
ROS Parameters

- Parameters can be set on the command line using *rosparam set*:



```
student@ros-vm:~$ rosparam set my_param 55.5
```

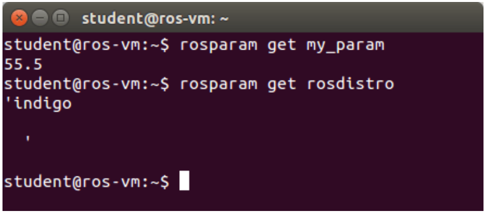
- List all the parameters currently on the ROS parameter server using *rosparam list*:



```
student@ros-vm:~$ rosparam list
/my_param
/rosdistro
/roslaunch/uris/host_ros_vm__33900
/rosversion
/run_id
student@ros-vm:~$
```

ROS Parameters

- Display the value of a parameter using *rosparam get*:



```
student@ros-vm:~$ rosparam get my_param
55.5
student@ros-vm:~$ rosparam get rosdistro
'indigo

  '

student@ros-vm:~$ 
```

- Parameters are accessed in code using a node handle.

ROS Parameters

- The *param()* node handle method is used to retrieve a parameter's value from the server and assign it to a program variable. If the specified parameter doesn't exist, this method assigns a default value.

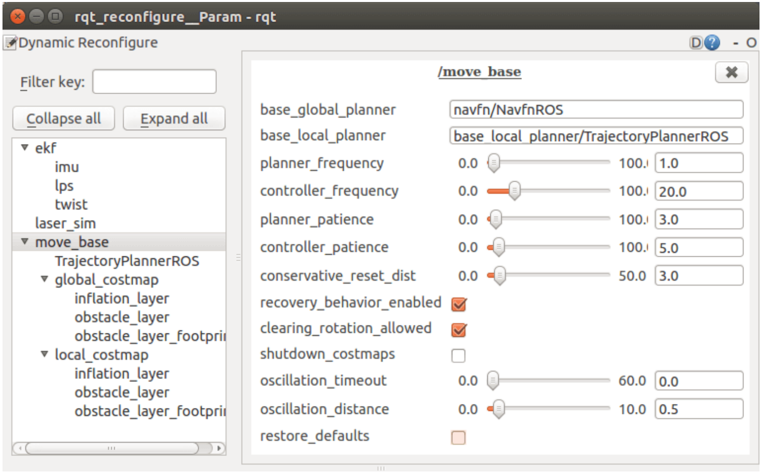
```
double var;
node_handle.param("param_name", var, 10.0);
```

- The *getParam()* method is the same as *param()*, except that it returns a boolean flag indicating the existence of the parameter, instead of setting the variable to a default value.

```
double var;
bool found = node_handle.getParam("param_name", var);
```

Dynamically Configurable Parameters

- Using *dynamic_reconfigure*, parameters can be defined that are capable of being dynamically adjusted at runtime.



rqt_reconfigure__Param - rqt

Dynamic Reconfigure

Filter key:

Collapse all | Expand all

```
▼ ekf
    imu
    lps
    twist
  laser_sim
▼ move_base
    TrajectoryPlannerROS
  ▼ global_costmap
      inflation_layer
      obstacle_layer
      obstacle_layer_footpri
  ▼ local_costmap
      inflation_layer
      obstacle_layer
      obstacle_layer_footpri
```

/move_base

base_global_planner	navfn/NavfnROS		
base_local_planner	base_local_planner/TrajectoryPlannerROS		
planner_frequency	0.0	100.(	1.0
controller_frequency	0.0	100.(	20.0
planner_patience	0.0	100.(	3.0
controller_patience	0.0	100.(	5.0
conservative_reset_dist	0.0	50.0	3.0
recovery_behavior_enabled	✓		
clearing_rotation_allowed	✓		
shutdown_costmaps	☐		
oscillation_timeout	0.0	60.0	0.0
oscillation_distance	0.0	10.0	0.5
restore_defaults	☐		

Dynamically Configurable Parameters

- ▷ The arguments of a dynamic parameter are:
  - > *name* – String that specifies the parameter's name in the ROS system, as well as the C++ structure variable name.
  - > *type* – Defines the type of parameter, which governs the type of graphical control that is shown in the GUI, as well as how it is represented in programs.
  - > *level* – Usually set to 0.
  - > *description* – A string that describes what the parameter is.
  - > *default* – The default value of the parameter.
  - > *min* – The minimum value of the parameter.
  - > *max* – The maximum value of the parameter.

Dynamically Configurable Parameters

- Dynamic parameters are defined in a *cfg* file. All *cfg* files are usually placed in a folder called 'cfg' in the root of the package.

- *cfg* files are actually Python programs:

```
#! /usr/bin/env python
PACKAGE='reconfig_example'

from dynamic_reconfigure.msg import SensorLevels
from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()
```

Dynamically Configurable Parameters

```
option_list =
    gen.enum(
      [gen.const("Option_1", int_t, 0, "A drop-down option"),
      gen.const("Option_2", int_t, 1, "A drop-down option"),
      gen.const("Option_3", int_t, 2, "A drop-down option")],
      "Different drop-down selections"
    )

gen.add("enable", bool_t, 0, "Boolean parameter", False)
gen.add("x", double_t, 0, "Floating point parameter", 0.0,
    0.0, 100.0)
gen.add("y", double_t, 0, "Floating point parameter", 0.0,
    -1.0, 1.0)
gen.add("list", int_t, 0, "List of options", 0, 0,
    2, edit_method=option_list)

exit(gen.generate(PACKAGE, PACKAGE, "Example"))
```

Dynamically Configurable Parameters

- To use dynamically configurable parameters in code, include the header files for *dynamic_reconfigure* server and the particular dynamic parameter header file generated from the *cfg* file:

```
#include <dynamic_reconfigure/server.h>
#include <reconfig_example/ExampleConfig.h>
```

- The name of the dynamic parameter header file is the name of the *cfg* file, with "Config" added to it.
- However, in order to generate the dynamic parameter header file, modifications must be made to *CMakeLists.txt*.

Dynamically Configurable Parameters

- Somewhere before the *catkin_package* line of *CMakeLists.txt*, this needs to be placed:

```
    generate_dynamic_reconfigure_options(cfg/Example.cfg)
```

- The path to the *cfg* file must be specified relative to the package root.

- Additionally, any node that uses a *dynamic_reconfigure* server will need the following *add_dependencies* line:

```
add_dependencies(reconfig_example ${PROJECT_NAME}_gencfg)
```

Dynamically Configurable Parameters

- First, define a callback function to be called whenever a dynamic parameter is changed:

```
void reconfig(reconfig_example::ExampleConfig& config,
                                           uint32_t level)
{
  // Code goes here
}
```

- Initializing a *dynamic_reconfigure* server can be done with two lines of code in the main function:

```
  dynamic_reconfigure::Server<reconfig_example::ExampleConfig> srv;
  srv.setCallback(boost::bind(reconfig, _1, _2));
```

- The first line instantiates the server, and the second line binds the *reconfig* callback function to the server object.

Dynamically Configurable Parameters

▷ The *config* argument is a structure containing the current values of each parameter defined in the *cfg* file.

▷ The *level* argument is usually unused.

▷ The callback function is called whenever a parameter is changed in the reconfigure GUI.

▷ The callback is also called once at start-up when it is assigned to the *dynamic_reconfigure* server in the main function.

Dynamically Configurable Parameters

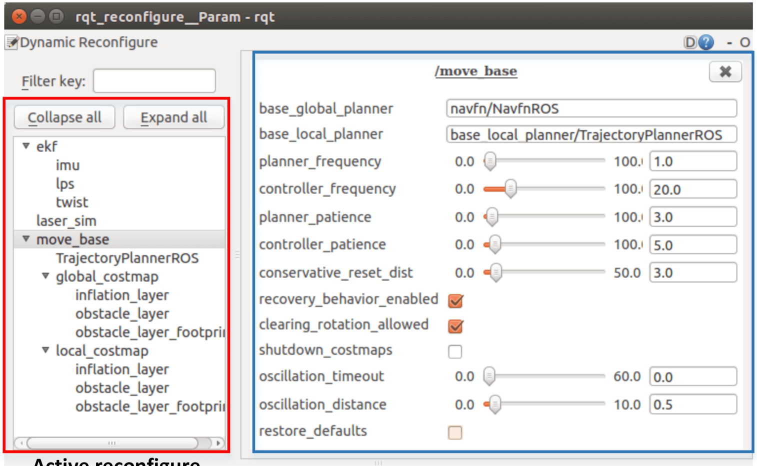
- After compiling and running any nodes with *dynamic_reconfigure* servers, open the GUI by opening a terminal and typing:

```
rosrun rqt_reconfigure rqt_reconfigure
```

- The parameters can be changed by sliding slider bars, editing text boxes, checking checkboxes, etc.

- Every time a parameter is changed, the corresponding server's reconfigure callback is called with the new set of parameters.

Dynamically Configurable Parameters



base_global_planner	navfn/NavfnROS		
base_local_planner	base_local_planner/TrajectoryPlannerROS		
planner_frequency	0.0	100.(	1.0
controller_frequency	0.0	100.(	20.0
planner_patience	0.0	100.(	3.0
controller_patience	0.0	100.(	5.0
conservative_reset_dist	0.0	50.0	3.0
recovery_behavior_enabled	✓		
clearing_rotation_allowed	✓		
shutdown_costmaps	☐		
oscillation_timeout	0.0	60.0	0.0
oscillation_distance	0.0	10.0	0.5
restore_defaults	☐		

Active reconfigure servers

Parameters for selected server

Setting and Loading Parameters in Launch Files

- Specific parameters can be set directly in the *node* tag:

```
<node pkg="package_name" type="node_type" name="node_name" >
  <param name="param_name" value="param_value" />
</node>
```

  - This method is used when it makes sense to set parameters specifically for a given launch file.

- Multiple parameter values can be loaded from a YAML file:

```
<node pkg="package_name" type="node_type" name="node_name" >
  <rosparam file="{path to file}/param_file.yaml" />
</node>
```

  - This is helpful when the same set of parameters are used in many different launch files.

Setting and Loading Parameters in Launch Files

- A simple YAML file looks like this:

```
float_param: 4.5
int_param: 7
string_param: hello_world
bool_param: true
```

- In this example, a launch file can load this YAML file to assign the given values to the four parameters named *float_param*, *int_param*, *string_param* and *bool_param*.