Dynamic Reconfigure

MRE/EME 5983 Robot Operating Systems

Dynamic Reconfigure

- The dynamic_reconfigure package provides a means to update node parameters at runtime without having to restart the node or implement a service based request to re-query the parameter server
- Dynamic reconfigure can update node parameter variables from the command line or through a ROS GUI
- Dynamic reconfigure parameters are defined for a given node

Dynamic Reconfigure Parameter Types

- ROS dynamic reconfigure parameters can have the following types
  - Boolean
  - Integer
  - Double
  - String
  - Enumerated list

Example

- Create a node that publishes two messages
  - Message 1: floating point computation

$$\text{msg.data} = A\ \cos(2\pi f t)$$

  - where,
    - A = is the trigonometric function amplitude (trig function = sin, cos or tan)
    - f = the trigonometric function frequency
    - t = time of the ROS system

  - Message 2: string entered by the user

- Require A, f, the trigonometric function and the string to be modified dynamically

Process For Creating Dynamic Reconfigure Variables

1. Create a configuration file defining the variables
2. Modify the CMakeLists.txt file to reflect the configuration file
3. Modify the node source to interface with the parameter server
4. Rebuild the package

Step 1: Configuration File

PublishNodeDynCfg.cfg

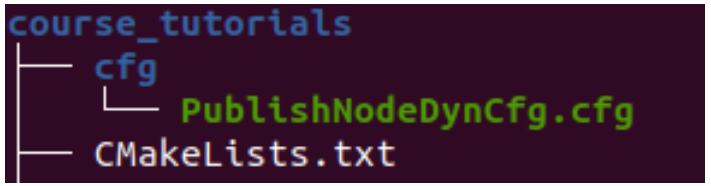
```
#!/usr/bin/env python
PACKAGE = "course_tutorials"

from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()

gen.add("bool_param",   bool_t,   0, "A Boolean parameter",    True)
gen.add("int_param",    int_t,    0, "An Integer parameter",  50, 0, 100)
gen.add("double_param", double_t, 0, "A double parameter",    .5, 0,   1)
gen.add("str_param",    str_t,    0, "A string parameter",    "Hello World")

func_enum = gen.enum([ gen.const("sin",      int_t, 0, "Sine function"),
                       gen.const("cos",      int_t, 1, "Cosine function"),
                       gen.const("tan",      int_t, 2, "Tangent function") ],
                     "An enum to set a function")

gen.add("fun_type", int_t, 0,
        "enumerated trig function", 1, 0, 2, edit_method=func_enum)

exit(gen.generate(PACKAGE, "course_tutorials", "PublishNodeDynCfg"))
```

- The dynamic reconfigure variables are defined in a configuration file
- Each dynamic reconfiguration parameter is added with a gen.add function call
- **This file is read and executed during the package complication, not at run time**

gen.add(name, paramtype, level, description, default val, min val, max val)

- Paramtype = bool_t, int_t, double_t or str_t
- Can create enumerated types as shown above

.cfg Location



Step 2: Modify the CMakeLists.txt

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(course_tutorials)
add_compile_options(-std=c++11)

# Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  dynamic_reconfigure
)

# Declare ROS dynamic reconfigure parameters
generate_dynamic_reconfigure_options(
  cfg/PublishNodeDynCfg.cfg
)
```

- Required CMakeLists.txt updates
  - dynamic_reconfigure dependency
  - generate_dynamic_reconfigure_options
- This will direct CMake to build the required C++ header files and Python include files to interface with the ROS parameter server

Step 3: Modify The Node Source Code

publisher_dyn.py

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

import rospy
import numpy as np
import sys
from std_msgs.msg import Float32
from std_msgs.msg import String

from dynamic_reconfigure.server import Server
from course_tutorials.cfg import PublishNodeDynCfgConfig
```

<package_name>.cfg

<configuration file base name + 'Config>

```
#################
# Main function
#################
if __name__ == '__main__':

    # Initialize the node and name it.
    rospy.init_node('publisher_py_node')
    rospy.loginfo('publisher_py_node running!')

    # Start node
    try:
        PublishNode()
    except rospy.ROSInterruptException:
        pass
```

publisher_dyn.py

```
class PublishNode():
    def __init__(self):
        """Publishing node to demonstrate dynamic reconfigure"""

        # Dynamic reconfigure variables
        self.dyn_config = []
        self.dyn_reconfig_bool = 'False'
        self.dyn_reconfig_int = 0
        self.dyn_reconfig_double = 0.0
        self.dyn_reconfig_string = ''
        self.dyn_reconfig_enum = 0

        # ROS dynamic reconfigure server
        self.srv = Server(PublishNodeDynCfgConfig, self.dyn_reconfig_callback)

        # ROS Topic Publisher
        self.pub_float = rospy.Publisher('float_msg', Float32, queue_size=10)
        self.pub_str = rospy.Publisher('str_msg', String, queue_size=10)

        # Define ROS rate
        self.rate = rospy.Rate(10)

        # Start ROS loop
        while not rospy.is_shutdown():

            #  Call publishers
            if( self.dyn_reconfig_bool ):
                self.publish_float_message()
                self.publish_str_message()

            # Control time step
            self.rate.sleep()

        return
```

Step 3: Modify The Node Source Code

publisher_dyn.py

In class PublishNode()…

```
################################
# Dynamic Reconfigure callback
################################
def dyn_reconfig_callback(self, config, level):
    self.dyn_reconfig_bool = config['bool_param']
    self.dyn_reconfig_int = config['int_param']
    self.dyn_reconfig_double = config['double_param']
    self.dyn_reconfig_string = config['str_param']
    self.dyn_reconfig_enum = config['fun_type']
    return config


############################
# Publish string messsage
############################
def publish_str_message(self):

    # Define message
    msg = String()
    msg.data = self.dyn_reconfig_string

    # Publish message
    self.pub_str.publish(msg)

    return
```

publisher_dyn.py

In class PublishNode()…

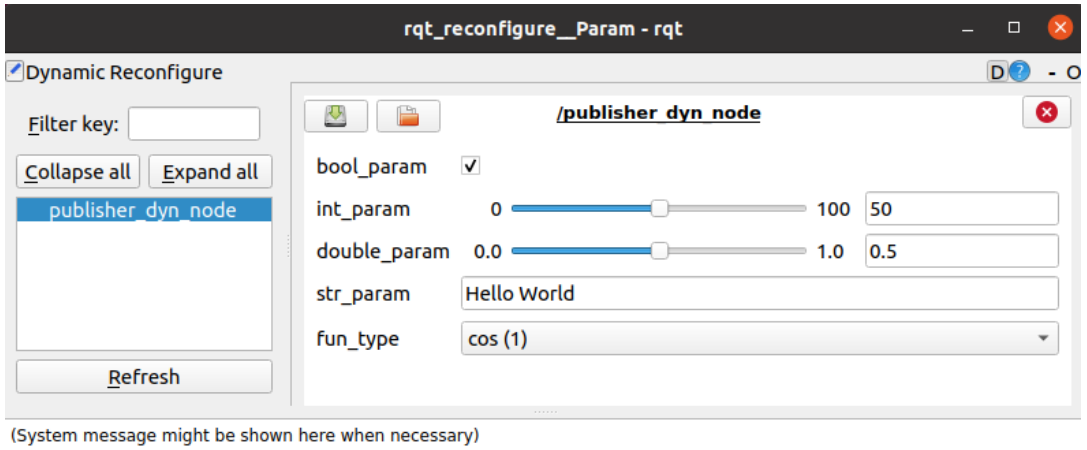
```
############################
# Publish float messsage
############################
def publish_float_message(self):

    # Get the ROS time and get function amplitude and independent variable
    t = rospy.get_time()
    f = self.dyn_reconfig_double
    x_val = 2*np.pi*f*t
    Amp = self.dyn_reconfig_int

    # Compute function
    if( self.dyn_reconfig_enum == 0 ):
        val = Amp*np.sin(x_val)
    elif( self.dyn_reconfig_enum == 1 ):
        val = Amp*np.cos(x_val)
    elif( self.dyn_reconfig_enum == 2 ):
        val = Amp*np.tan(x_val)
    else:
        rospy.logerr('Invalid enumerated type')
        sys.exit(0)

    # Define message
    msg = Float32()
    msg.data = val

    # Publish message
    self.pub_float.publish(msg)

    return
```

Step 4: Rebuild package

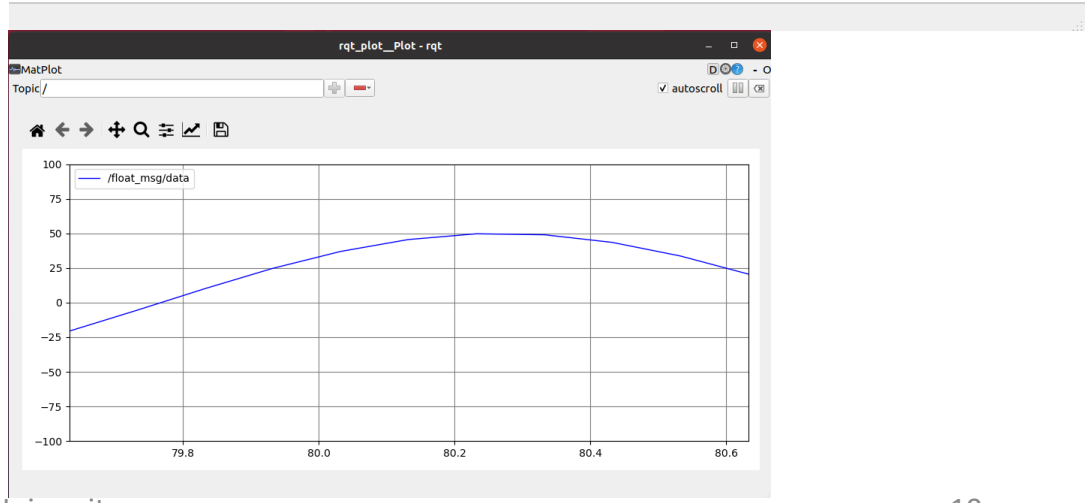
dynamic_reconfigure.launch

```
<launch>

  <!-- Publisher node -->
  <node
       pkg = "course_tutorials"
       type = "publisher_dyn.py"
       name = "publisher_dyn_node"
       />

  <!-- Dynamic reconfigure -->
  <node
       pkg = "rqt_reconfigure"
       type = "rqt_reconfigure"
       name = "rqt_console"
       args = "-t"
       />

  <!-- rqt_plot -->
  <node
       pkg = "rqt_plot"
       type = "rqt_plot"
       name = "rqt_plot_node"
       />

</launch>
```

- Package rebuild and source devel/setup.bash execute
  - publisher_dyn_node
  - rqt_reconfigure

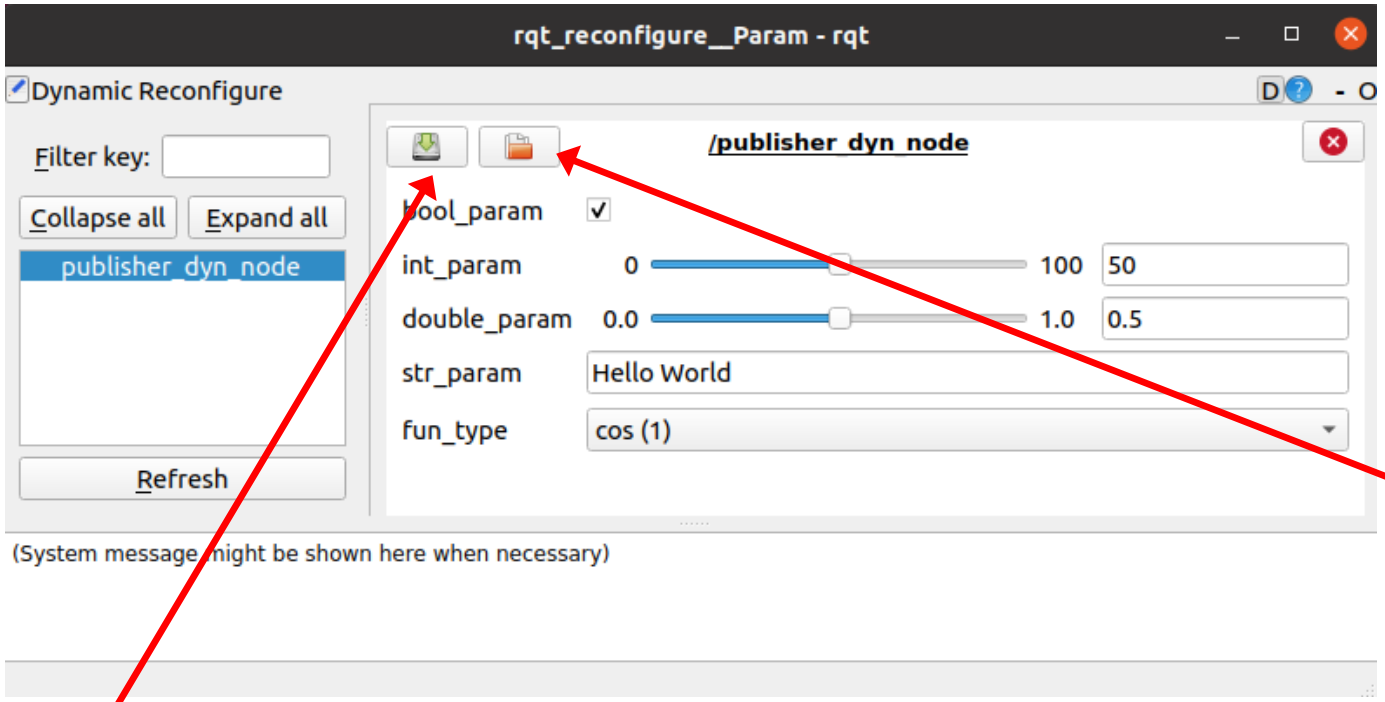


  - rqt_plot



Saving and Loading Parameters

- We can save and load parameter through the GUI



Load parameters

Save parameters: Save the file with a .yaml extension

Configuration Files Format

params.yaml

```
!!python/object/new:dynamic_reconfigure.encoding.Config
dictitems:
  bool_param: true
  double_param: 0.5
  fun_type: 1
  groups: !!python/object/new:dynamic_reconfigure.encoding.Config
    dictitems:
      bool_param: true
      double_param: 0.5
      fun_type: 1
      groups: !!python/object/new:dynamic_reconfigure.encoding.Config
        state: []
      id: 0
      int_param: 50
      name: Default
      parameters: !!python/object/new:dynamic_reconfigure.encoding.Config
        state: []
      parent: 0
      state: true
      str_param: Hello World
      type: ''
    state: []
  int_param: 50
  str_param: Hello World
state: []
```

params_short.yaml

```
bool_param: true
double_param: 0.75
int_param: 25
str_param: Hello World Version 2
fun_type: 0
```

Parameter files can be stored in the params directory under the package directory. To load a .yaml file at launch:

<node name="dynamic_reconfigure_node" pkg="dynamic_reconfigure" type="dynparam“
args="load /publisher_dyn_node $(find course_tutorials)/params/params_short.yaml"/>

Summary

- Dynamic Reconfigure is a very convenient method of changing node parameters at run time

- As we begin to solve more complicated ROS problems with many parameters, Dynamic Reconfigure will make us much more efficient