

Robot Operating System (ROS2)

Vilma Muço

Université de Toulon, Master Ingénierie des Systèmes Complexes (ISC) -Erasmus
Mundus MIR

November, 2023

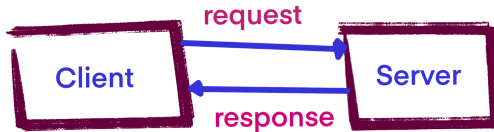


Table of Contents

- Services
- Python code
- Actions

Services

- ▶ The node that sends a request is called the client node
- ▶ The node that provides the response is the server node
- ▶ The structure of the request and the response is defined in a .srv file



Turtlesim

```
$ros2 service list
$ros2 service type /turtle1/set_pen
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service list
/clear
/kill
/reset
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service type /turtle1/set_pen
turtlesim/srv/SetPen
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 interface show turtlesim/srv/SetPen
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

Ros Help command

```
$ros2 -h
```

```
$ros2 service -h
```

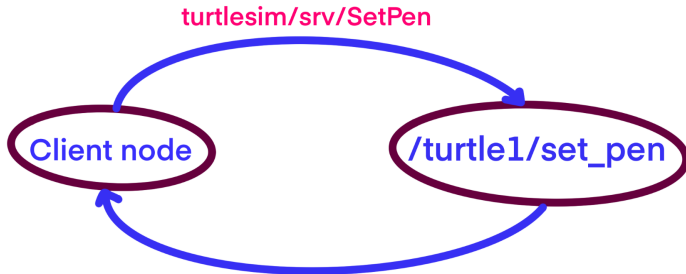
```
vilma@:~$ ros2 service -h
usage: ros2 service [-h] [--include-hidden-services] Call `ros2 service <command> -h` for more details

Various service related sub-commands

options:
  -h, --help            show this help message and exit
  --include-hidden-services
                        Consider hidden services as well

Commands:
  call    Call a service
  find    Output a list of available services of a given type
  list    Output a list of available services
  type    Output a service's type

Call `ros2 service <command> -h` for more detailed usage.
vilma@:~$
```



Creating a new package

```
ros2 pkg create --build-type ament_python --dependencies rclpy p_client_server
```

- Modify package.xml (add exec_depend for turtlesim) and setup.py
- Create the clientNode.py file
- Add an entry point for the client node

```
p_client_server > setup.py > ...
1 from setuptools import find_packages, setup
2
3 package_name = 'p_client_server'
4
5 setup(
6     name=package_name,
7     version='0.0.0',
8     packages=find_packages(exclude=['test']),
9     data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='vilma',
17     maintainer_email='vilma.muco@gmail.com',
18     description='TODO: Package description',
19     license='Apache License 2.0',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'client_node = p_client_server.clientNode:main',
24         ],
25     },
26 )
27
```

Client Program

```
1 import rclpy
2 from rclpy.node import Node
3 from turtlesim.srv import SetPen
4 import sys
5 import time
6
7 class ClientNode(Node):
8
9     def __init__(self):
10         super().__init__('client_node')
11         self.cli = self.create_client(SetPen, '/test_namespace/turtle1/set_pen')
12         while not self.cli.wait_for_service(timeout_sec=3.0):
13             self.get_logger().info('service not available, waiting again...')
14         self.req = SetPen.Request()
15
16     def send_request(self, red, green, blue):
17         self.req.r = red
18         self.req.g = green
19         self.req.b = blue
20         self.req.width = 5
21         self.req.off = 0
22         self.future = self.cli.call_async(self.req)
23         rclpy.spin_until_future_complete(self, self.future)
24
25 def main():
26     rclpy.init()
27
28     red = 255 # pen rgb colors
29     green = 0
30     blue = 0
31     client_node = ClientNode()
32
33     while True:
34         # switch between red and green
35         if red == 255:
36             red = 0
37         else:
38             red = 255
39
40         if green == 255:
41             green = 0
42         else:
43             green = 255
44
45         response = client_node.send_request(red, green, blue)
46         time.sleep(0.2)
47
48     client_node.destroy_node()
49     rclpy.shutdown()
50
51 if __name__ == '__main__':
52     main()
```


Code Analysis

```

1  import rclpy
2  from rclpy.node import Node
3  from turtlesim.srv import SetPen
4  import sys
5  import time
6
7  class ClientNode(Node):
8
9      def __init__(self):
10         super().__init__('client_node')
11         self.cli = self.create_client(SetPen, '/test_namespace/turtle1/set_pen')
12         while not self.cli.wait_for_service(timeout_sec=3.0):
13             self.get_logger().info('service not available, waiting again...')
14         self.req = SetPen.Request()
15

```

Code Analysis

```

1  def send_request(self, red, green, blue):
2      self.req.r = red
3      self.req.g = green
4      self.req.b = blue
5      self.req.width = 5
6      self.req.off = 0
7      self.future = self.cli.call_async(self.req)
8      rclpy.spin_until_future_complete(self, self.future)
9
10 def main():
11     rclpy.init()
12     red = 255 # pen rgb colors
13     green = 0
14     blue = 0
15     client_node = ClientNode()
16
17     # switch between red and green
18     while True:
19         red, green = green, red
20         response = client_node.send_request(red, green, blue)
21         time.sleep(0.2)
22
23     client_node.destroy_node()
24     rclpy.shutdown()
25
26 if __name__ == '__main__':
27     main()

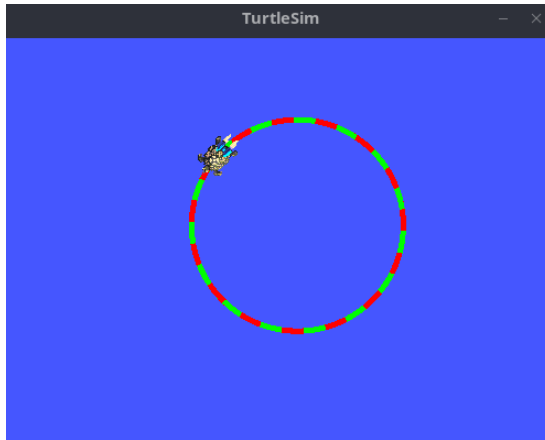
```

Run the program

- ▶ Launch the circle_path_launch.py file
- ▶ Build and run the client node

```
colcon build --packages-select p_client_server  
ros2 run p_client_server client_node
```

Run the program



Custom message and service types

- ▶ Good practice to use predefined interface definitions
- ▶ For the moment there is no way of generating a .msg and a .srv file in a pure Python package
- ▶ We can only define new interfaces in a CMake package
- ▶ CMake is an open source, cross-platform family of tools designed to build, test, and package software.
- ▶ CMake gives you control of the software compilation process using simple independent configuration files.
- ▶ We can use `ament_cmake_python` to build interfaces and python code in the same package

- It is good practice to keep .msg and .srv files in their own directories within a package

```
$ros2 pkg create --build-type ament_cmake p_interfaces  
$cd p_interfaces  
$mkdir msg  
$mkdir srv
```

```
vilma@vilma-Precision-3571:~/ros2_ws/p_interfaces$ mkdir srv  
vilma@vilma-Precision-3571:~/ros2_ws/p_interfaces$ mkdir msg  
vilma@vilma-Precision-3571:~/ros2_ws/p_interfaces$ ll  
total 32  
drwxrwxr-x 6 vilma vilma 4096 nov. 8 11:29 ./  
drwxrwxr-x 10 vilma vilma 4096 nov. 8 11:25 ../  
-rw-rw-r-- 1 vilma vilma 905 nov. 8 11:25 CMakeLists.txt  
drwxrwxr-x 3 vilma vilma 4096 nov. 8 11:25 include/  
drwxrwxr-x 2 vilma vilma 4096 nov. 8 11:29 msg/  
-rw-rw-r-- 1 vilma vilma 603 nov. 8 11:25 package.xml  
drwxrwxr-x 2 vilma vilma 4096 nov. 8 11:25 src/  
drwxrwxr-x 2 vilma vilma 4096 nov. 8 11:29 srv/  
vilma@vilma-Precision-3571:~/ros2_ws/p_interfaces$
```

- ▶ Create a new file Num.msg that transfers a single 64-bit integer called num:

```
int64 num
```

- ▶ Create a new file ChangeRadius.srv with the following request and response structure:

```
int64 radius
---
bool radius_changed
geometry_msgs/Vector3 linear_velocity
```

► Modify the CMakeLists.txt:

```
find_package(geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(\${PROJECT_NAME}
  "msg/Num.msg"
  "srv/ChangeRadius.srv"
  DEPENDENCIES geometry_msgs # Add packages that above msg and srv
                                     # depend on, in this case geometry_msgs
)
```


Package.xml

- ▶ Because the interfaces rely on `rosidl_default_generators` for generating language-specific code, you need to declare a build tool dependency on it
- ▶ `rosidl_default_runtime` is a runtime or execution-stage dependency, needed to be able to use the interfaces later
- ▶ `rosidl_interface_packages` is the name of the dependency group that your package, should be associated with, using the `member_of_group` tag.

```
<depend>geometry_msgs</depend>
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

Build the p_interfaces package

```
colcon build --packages-select p_interfaces
source install/setup.bash
ros2 interface show p_interfaces/msg/Num
ros2 interface show p_interfaces/srv/ChangeRadius
```

```
vilmagvilma-Precision-3571:~/ros2_ws$ colcon build --packages-select p_interfaces
Starting >>> p_interfaces
Finished <<< p_interfaces [1.12s]

Summary: 1 package finished [1.21s]
vilmagvilma-Precision-3571:~/ros2_ws$ source install/setup.bash
vilmagvilma-Precision-3571:~/ros2_ws$ ros2 interface show p_interfaces/msg/Num
int64 num
vilmagvilma-Precision-3571:~/ros2_ws$ ros2 interface show p_interfaces/srv/ChangeRadius
int64 radius
---
bool radius_changed
geometry_msgs/Vector3 linear_velocity
  float64 x
  float64 y
  float64 z
vilmagvilma-Precision-3571:~/ros2_ws$
```

Test the new Interface

In order to test the new Interface:

- Add a service to the circleController node to change the radius of the trajectory of the turtle.

Test the new Interface

```

1  import rclpy
2  from rclpy.node import Node
3  from geometry_msgs.msg import Twist
4
5  from p_interfaces.srv import ChangeRadius
6
7
8  class CircleController(Node):
9      def __init__(self):
10         super().__init__('circle_controller')
11         self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
12         timer_period = 0.1 # seconds
13         self.timer = self.create_timer(timer_period, self.timer_callback)
14         self.srv = self.create_service(ChangeRadius, 'change_radius', self.change_radius_callback)
15         self.radius = 1.0
16

```

Test the new Interface

```

1  def timer_callback(self):
2      vel = Twist()
3      vel.linear.x = self.radius * 1.0  # linear velocity = radius * angular velocity
4      vel.angular.z = 1.0
5      self.publisher_.publish(vel)
6
7  def change_radius_callback(self, request, response):
8      self.radius = request.radius
9      response.linear_velocity.x = self.radius * 1.0
10     response.linear_velocity.y = 0.0
11     response.linear_velocity.z = 1.0
12     response.success = True
13     return response
14

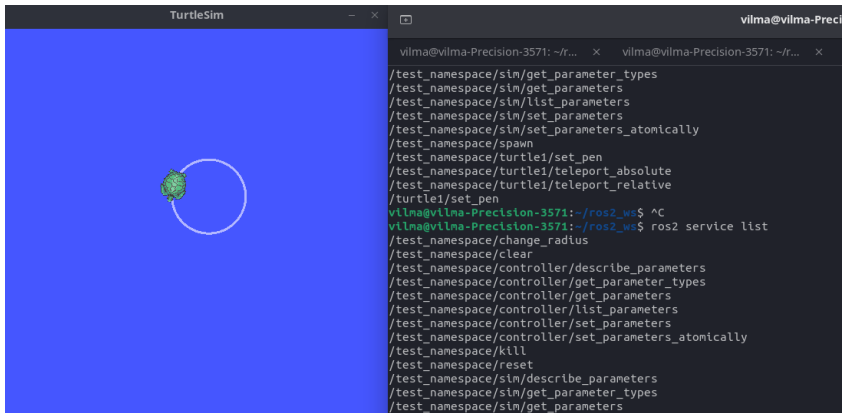
```

Build the code

- ▶ Modify the package.xml
- ▶ Build the package

```
<exec_depend>p_interfaces</exec_depend>
colcon build --packages-select p_controller
source install/setup.bash
ros2 launch p_controller/p_controller/launch/circle_path_launch.py
```

Build the code



ros2 service call

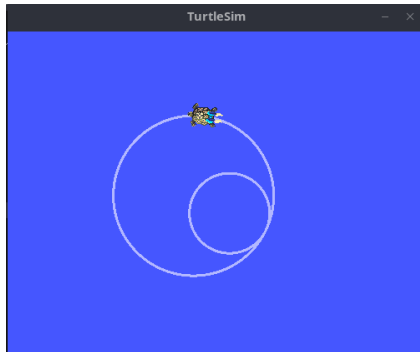
- We can use the command line to request a service

```
$ros2 service call <service_name> <service_type> <arguments>
$ros2 service call /test_namespace/change_radius p_interfaces/srv/ChangeRadius "{radius : 2}"
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service call /test_namespace/change_radius p_interfaces/srv/ChangeRadius "{radius : 2}"
waiting for service to become available...
requester: making request: p_interfaces.srv.ChangeRadius_Request(radius=2)

response:
p_interfaces.srv.ChangeRadius_Response(radius_changed=True, linear_velocity=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=1.0))
vilma@vilma-Precision-3571:~/ros2_ws$
```

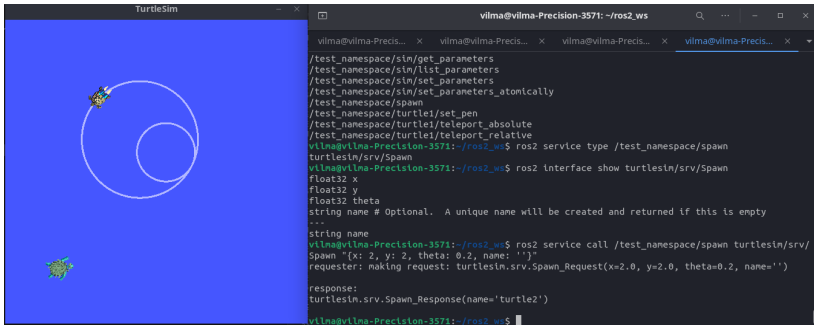

ros2 service call



ros2 service call

► Lets spawn another turtle

```
1 $ros2 service list
2 $ros2 service type /test_namespace/spawn
3 $ros2 interface show turtlesim/srv/Spawn
4 $ros2 service call /test_namespace/spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: ''}"
```



The screenshot shows a ROS2 environment. On the left is the TurtleSim window, which displays a blue field with two turtles. One turtle is at the bottom left, and another is at the top left, with their trajectories shown as white lines. On the right is a terminal window showing the execution of ROS2 service commands. The commands are as follows:

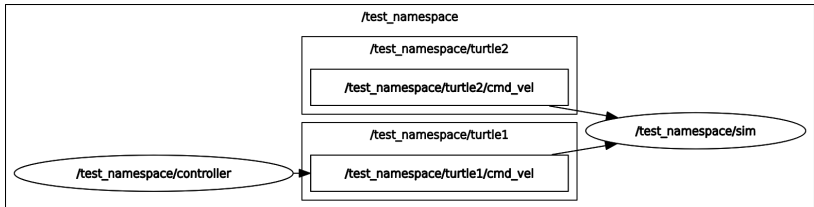
```
vilma@vilma-Precision-3571: ~/ros2_ws
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service list
/test_namespace/sin/get_parameters
/test_namespace/sin/list_parameters
/test_namespace/sin/set_parameters
/test_namespace/sin/set_parameters_atomically
/test_namespace/spawn
/test_namespace/turtle1/set_pen
/test_namespace/turtle1/teleport_absolute
/test_namespace/turtle1/teleport_relative
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service type /test_namespace/spawn
turtlesim/srv/Spawn
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 interface show turtlesim/srv/Spawn
float32 x
float32 y
float32 theta
string name # Optional. A unique name will be created and returned if this is empty
...
string name
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service call /test_namespace/spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: ''}"
requester: making request: turtlesim.srv.Spawn_Request(x=2.0, y=2.0, theta=0.2, name='')
response:
turtlesim.srv.Spawn_Response(name='turtle2')
vilma@vilma-Precision-3571:~/ros2_ws$
```

ros2 service call

► Pay attention to the namespaces

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 service list
/test_namespace/change_radius
/test_namespace/clear
/test_namespace/controller/describe_parameters
/test_namespace/controller/get_parameter_types
/test_namespace/controller/get_parameters
/test_namespace/controller/list_parameters
/test_namespace/controller/set_parameters
/test_namespace/controller/set_parameters_atomically
/test_namespace/kill
/test_namespace/reset
/test_namespace/sim/describe_parameters
/test_namespace/sim/get_parameter_types
/test_namespace/sim/get_parameters
/test_namespace/sim/list_parameters
/test_namespace/sim/set_parameters
/test_namespace/sim/set_parameters_atomically
/test_namespace/spawn
/test_namespace/turtle1/set_pen
/test_namespace/turtle1/teleport_absolute
/test_namespace/turtle1/teleport_relative
/test_namespace/turtle2/set_pen
/test_namespace/turtle2/teleport_absolute
/test_namespace/turtle2/teleport_relative
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 topic list
parameter_events
/rosout
/test_namespace/turtle1/cmd_vel
/test_namespace/turtle1/color_sensor
/test_namespace/turtle1/pose
/test_namespace/turtle2/cmd_vel
/test_namespace/turtle2/color_sensor
/test_namespace/turtle2/pose
vilma@vilma-Precision-3571:~/ros2_ws$
```

ros2 service call



Remapping

- ▶ Remapping allows you to reassign default node properties, like node name, topic names, service names, etc., to custom values.
- ▶ Remapping through the command line tool

```
ros2 run my_package node_executable --ros-args ...  
ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
```

- ▶ Launch files

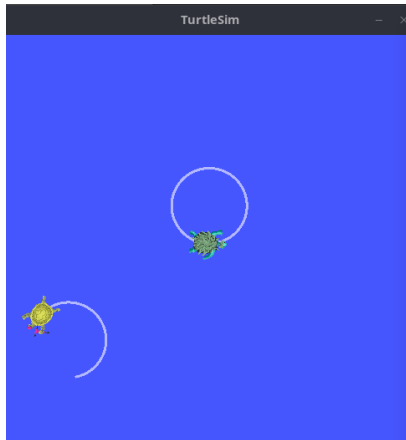
Remapping

► Modify the circle_path_launch.py

```
1  from launch import LaunchDescription
2  from launch_ros.actions import Node
3
4  def generate_launch_description():
5      return LaunchDescription([
6          Node(
7              package='turtlesim',
8              namespace='test_namespace',
9              executable='turtlesim_node',
10             name='sim',
11             remappings=[
12                 ('/test_namespace/turtle2/cmd_vel', '/test_namespace/turtle1/cmd_vel'),
13             ]
14         ),
15         Node(
16             package='p_controller',
17             namespace='test_namespace',
18             executable='circle_controller',
19             name='controller',
20         )
21     ])
```

Remapping

- ▶ Launch the nodes and spawn another turtle



Using parameters

- ▶ Adding parameters that can be configured during runtime
- ▶ Ex. ip address and port number for a Gps Receiver
- ▶ We can modify the value of the parameter :
 - ▶ via the console
 - ▶ from a launch file

```
$ros2 param list
```

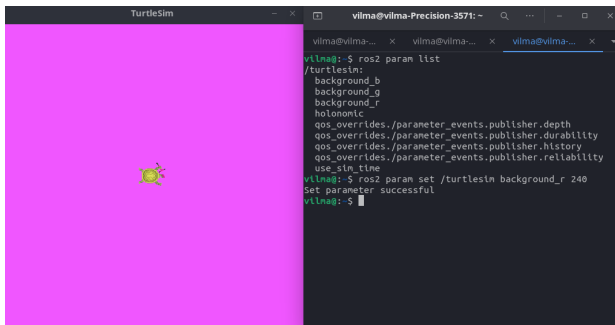
```

vilm@vilm:~$ ros2 param list
/ackermann
/background_3
/background_5
/background_7
/background
/parameter_events.publisher.depth
/parameter_events.publisher.dynamicity
/parameter_events.publisher.history
/parameter_events.publisher.reliability
/parameter_events.publisher.size_in_kb
vilm@vilm:~$
    
```


Using parameters

- Ex. change the background color of turtlesim

```
$ros2 param set /turtlesim background_r 240
```



Using parameters

- Add the angular z velocity of the turtle as a parameter in the circle controller example

Modify the python code to use parameters

```
1  def __init__(self):
2      super().__init__('circle_controller')
3      self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
4      timer_period = 0.1 # seconds
5      self.timer = self.create_timer(timer_period, self.timer_callback)
6      self.srv = self.create_service(ChangeRadius, 'change_radius', self.change_radius_callback)
7      self.radius = 1.0
8      self.declare_parameter('angular_z_velocity', 1.0)
9
10 def timer_callback(self):
11     # if the angular velocity param is bigger than 1.0, reset it to 1.0
12     if self.get_parameter('angular_z_velocity').value > 1.0:
13         new_param = rclpy.parameter.Parameter('angular_z_velocity', rclpy.Parameter.Type.DOUBLE, 1.0)
14         self.set_parameters([new_param])
15
16     vel = Twist()
17     vel.linear.x = self.radius * 1.0 # linear velocity = radius * angular velocity
18     vel.angular.z = self.get_parameter('angular_z_velocity').value
19
20     self.publisher_.publish(vel)
```

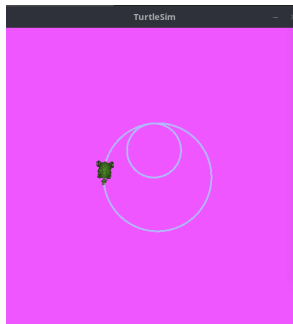
Build and Run

```
1 colcon build --packages-select p_controller
2 source install/setup.bash
3 $ros2 launch p_controller/p_controller/launch/circle_path_launch.py
4 $ros2 param list
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 param list
/test_namespace/controller:
  angular_z_velocity
  use_sim_time
/test_namespace/sim:
  background_b
  background_g
  background_r
  holonomic
  qos_overrides./parameter_events.publisher.depth
  qos_overrides./parameter_events.publisher.durability
  qos_overrides./parameter_events.publisher.history
  qos_overrides./parameter_events.publisher.reliability
  use_sim_time
vilma@vilma-Precision-3571:~/ros2_ws$ ^C
vilma@vilma-Precision-3571:~/ros2_ws$
```

Build and Run

```
ros2 param set /test_namespace/controller angular_z_velocity 0.5
```



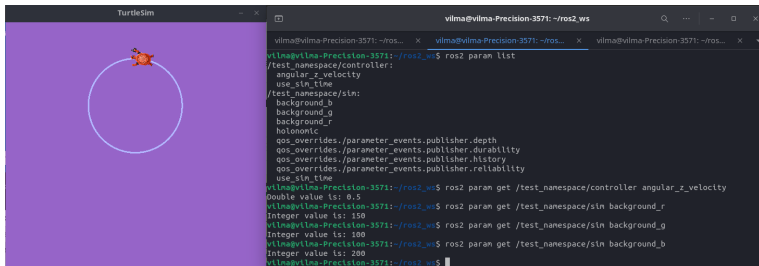
Param change via Launch files

```

1  from launch import LaunchDescription
2  from launch_ros.actions import Node
3
4  def generate_launch_description():
5      return LaunchDescription([
6          Node(
7              package='turtlesim',
8              namespace='test_namespace',
9              executable='turtlesim_node',
10             name='sim',
11             remappings=[('/test_namespace/turtle2/cmd_vel', '/test_namespace/turtle1/cmd_vel')],
12             parameters=[
13                 {"background_b": 200},
14                 {"background_g": 100},
15                 {"background_r": 150}
16             ]
17         ),
18         Node(
19             package='p_controller',
20             namespace='test_namespace',
21             executable='circle_controller',
22             name='controller',
23             parameters=[
24                 {"angular_z_velocity": 0.5}
25             ]
26         )
27     ])

```

Run the Launch file



```

vilma@vilma-Precision-3571: ~/ros2_ws
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 param list
/test_namespace/controller:
  angular_z_velocity
  use_sin_time
/test_namespace/sin:
  background_b
  background_g
  background_r
  holonomic
  qos_overrides./parameter_events.publisher.depth
  qos_overrides./parameter_events.publisher.durability
  qos_overrides./parameter_events.publisher.history
  qos_overrides./parameter_events.publisher.reliability
  use_sin_time
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 param get /test_namespace/controller angular_z_velocity
Double value is: 0.5
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 param get /test_namespace/sin background_r
Integer value is: 150
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 param get /test_namespace/sin background_g
Integer value is: 100
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 param get /test_namespace/sin background_b
Integer value is: 200
vilma@vilma-Precision-3571:~/ros2_ws$

```

- Modify the service example code and use the service `/turtle1/set_pen` to draw two circles with 2 different colors



Custom Actions

- ▶ We can custom-define actions in our packages
- ▶ Actions are defined in .action files
- ▶ A request message is sent from an action client to an action server initiating a new goal.
- ▶ A result message is sent from an action server to an action client when a goal is done.
- ▶ Feedback messages are periodically sent from an action server to an action client with updates about a goal.

```
# Request  
---  
# Result  
---  
# Feedback
```

Custom Actions

► Create a new package

```
cd ~/ros2_ws/src
ros2 pkg create --license Apache-2.0 p_action_interfaces
cd p_action_interfaces
mkdir action
vi GoToPoint.action
```

► Create an action with goal to reach a point and stop

```
1  # Request
2  float32 x
3  float32 y
4  ---
5  # Result
6  bool success
7  ---
8  # Feedback
9  turtlesim/Pose pose
```

Build the custom action

► Modify the CMakeLists.txt

```
find_package(turtlesim REQUIRED)

find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "action/GoToPoint.action"
  DEPENDENCIES turtlesim
)
ament_package()
```

► Add the required dependencies to our package.xml

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
<exec_depend>turtlesim</exec_depend>
```

Build the custom action

```
colcon build --packages-select p_action_interfaces  
source install/setup.bash  
ros2 interface show p_action_interfaces/action/GoToPoint
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 interface show p_action_interfaces/action/GoToPoint  
# Request  
float32 x  
float32 y  
---  
# Result  
bool success  
---  
# Feedback  
turtlesim/Pose pose  
float32 x  
float32 y  
float32 theta  
float32 linear_velocity  
float32 angular_velocity  
vilma@vilma-Precision-3571:~/ros2_ws$
```

Writing an action server

```
1
2 class GoToGoal(Node):
3     def __init__(self):
4         super().__init__("go_to_goal")
5         self.publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
6
7         velocity_callback_group = MutuallyExclusiveCallbackGroup()
8         self.subscription = self.create_subscription(
9             Pose,
10            "turtle1/pose",
11            self.pose_listener_callback,
12            10,
13            callback_group=velocity_callback_group,
14        )
15
16        action_callback_group = MutuallyExclusiveCallbackGroup()
17        self.action_server = ActionServer(
18            self,
19            GoToPoint,
20            "go_to_point",
21            self.action_callback,
22            callback_group=action_callback_group,
23        )
24
25        self.current_pose = Pose()
26
27        # in the beginning, the turtle is happy where it is until it receives new instructions
28        self.goal_reached = True
29        self.velocity = Twist() # 0 by default
30        self.goal_x = 0.0
31        self.goal_y = 0.0
32        self.EPSILON_ERROR = 0.1
```

Writing an action server

```
1 def action_callback(self, goal_handle):
2     self.get_logger().info('Executing goal...')
3     if not self.goal_reached:
4         print('got new goal before old one reached')
5         # set the goal
6         self.goal_x = goal_handle.request.x
7         self.goal_y = goal_handle.request.y
8         feedback_msg = GoToPoint.Feedback()
9         feedback_msg.pose = self.current_pose
10
11         # while the goal is not reached, send feedback to the client
12         diff_x = abs(self.current_pose.x - self.goal_x)
13         diff_y = abs(self.current_pose.y - self.goal_y)
14         while diff_x > self.EPSILON_ERROR or diff_y > self.EPSILON_ERROR:
15             print('sending feedback')
16             goal_handle.publish_feedback(feedback_msg)
17             feedback_msg.pose = self.current_pose
18             diff_x = abs(self.current_pose.x - self.goal_x)
19             diff_y = abs(self.current_pose.y - self.goal_y)
20             # put only this thread to sleep for 1 seconds
21             time.sleep(1)
22
23         goal_handle.succeed()
24         result = GoToPoint.Result()
25         result.success = True
26         self.goal_reached = True
27         return result
```

Writing an action server

```
1
2
3 def pose_listener_callback(self, msg):
4     self.current_pose = msg
5     angle = 0.0
6     delta = 0.0
7
8     diff_x = abs(self.current_pose.x - self.goal_x)
9     diff_y = abs(self.current_pose.y - self.goal_y)
10    # calculate the angle between the current pose and the goal pose
11    # and rotate the robot towards the goal
12    angle = math.atan2(self.goal_y - self.current_pose.y, self.goal_x - self.current_pose.x)
13
14    delta = angle - self.current_pose.theta
15    self.velocity.linear.x = 0.0
16
17    distance = math.sqrt(diff_x * diff_x + diff_y * diff_y)
18    if diff_x < self.EPSILON_ERROR and diff_y < self.EPSILON_ERROR:
19        self.velocity.angular.z = 0.0
20    elif delta < -0.02 or delta > 0.02:
21        self.velocity.angular.z = delta * 0.5
22    else:
23        self.velocity.angular.z = 0.0
24        self.velocity.linear.x = distance * 0.5
25
26    # once we have updated what we want to do, publish the velocity
27    if self.velocity.linear.x != 0.0 or self.velocity.angular.z != 0.0:
28        self.publisher_.publish(self.velocity)
```

Writing an action server

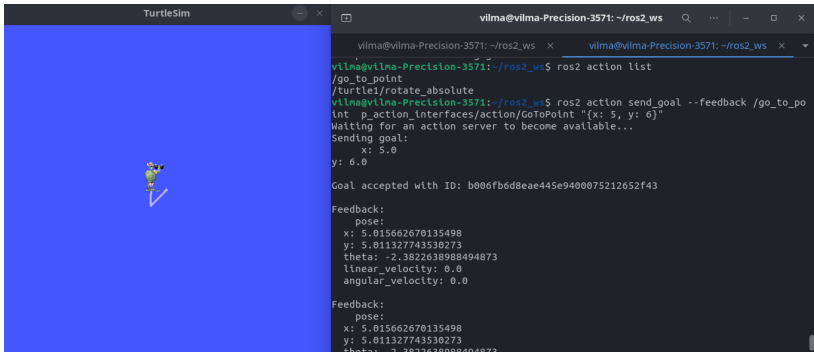
```
1 def main(args=None):
2     rclpy.init(args=args)
3
4     go_to_goal = GoToGoal()
5     executor = MultiThreadedExecutor() # needed to run functions concurrently
6     executor.add_node(go_to_goal)
7
8     executor.spin()
9
10    go_to_goal.destroy_node()
11    rclpy.shutdown()
12
13
14 if __name__ == '__main__':
15     main()
```


Run the program

- ▶ Run turtlesim node and the p_action_example package on two different terminals
- ▶ In a third terminal send a goal to the action via command line

```
colcon build --packages-select p_action_example && ros2 run p_action_example go_to_goal  
ros2 run turtlesim turtlesim_node  
ros2 action list  
ros2 action send_goal --feedback /go_to_point p_action_interfaces/action/GoToPoint "{x: 5, y: 6}"
```

Run the program



Action Client

- Create an action client that moves the turtle in a square

```
1 import rclpy
2 from rclpy.action import ActionClient
3 from rclpy.node import Node
4 import time
5 from p_action_interfaces.action import GoToPoint
6
7 class DrawSquareActionClient(Node):
8     def __init__(self):
9         super().__init__('draw_square_action_client')
10        self._action_client = ActionClient(self, GoToPoint, 'go_to_point')
11        # create list with 4 edges of the square in tuple format starting from coordinate (5, 5)
12        # and with distance of a units between each edge going counter-clockwise
13        a = 3.0
14        self.edges = [(5.0, 5.0), (a + 5.0, 5.0), (a + 5.0, a + 5.0), (5.0, a + 5.0), (5.0, 5.0)]
15        self.current_step = 0 # the current edge of the square the turtle is on
16
17    def send_goal(self):
18        x, y = self.edges[self.current_step]
19        goal_msg = GoToPoint.Goal()
20        goal_msg.x = x
21        goal_msg.y = y
22
23        self._action_client.wait_for_server()
```

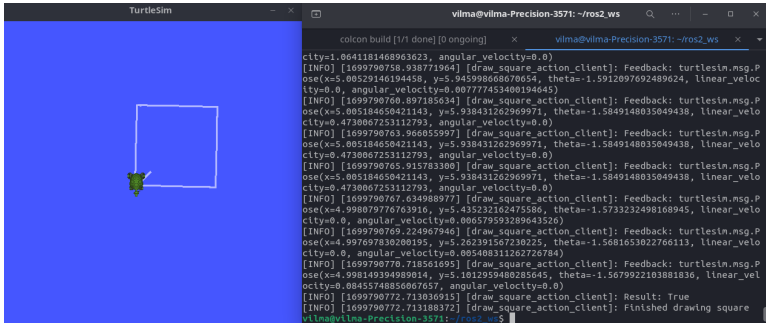
Action Client

```
1 def send_goal(self):
2     x, y = self.edges[self.current_step]
3     goal_msg = GoToPoint.Goal()
4     goal_msg.x = x
5     goal_msg.y = y
6
7     self._action_client.wait_for_server()
8     self._send_goal_future = self._action_client.send_goal_async(goal_msg,
9     feedback_callback=self.feedback_callback)
10    self._send_goal_future.add_done_callback(self.goal_response_callback)
11
12    def goal_response_callback(self, future):
13        goal_handle = future.result()
14        if not goal_handle.accepted:
15            self.get_logger().info('Goal rejected :(')
16            return
17
18        self.get_logger().info('Goal accepted :)')
19        self._get_result_future = goal_handle.get_result_async()
20        self._get_result_future.add_done_callback(self.get_result_callback)
```

Action Client

```
1  def get_result_callback(self, future):
2      result = future.result().result
3      self.get_logger().info('Result: {0}'.format(result.success))
4      # if result is True go to next step
5      if result.success:
6          self.current_step += 1
7          if self.current_step < len(self.edges):
8              self.send_goal()
9          else:
10             self.get_logger().info('Finished drawing square')
11             rclpy.shutdown()
12
13  def feedback_callback(self, feedback_msg):
14      self.get_logger().info('Feedback: {0}'.format(feedback_msg.feedback.pose))
15
16  def main(args=None):
17      rclpy.init(args=args)
18      action_client = DrawSquareActionClient()
19      action_client.send_goal()
20      rclpy.spin(action_client)
21
22  if __name__ == '__main__':
23      main()
```

Run the program



The screenshot displays two windows. On the left is the 'TurtleSim' window, which shows a green turtle on a blue background, having just completed drawing a white square. On the right is a terminal window titled 'vilma@vilma-Precision-3571: ~/ros2_ws'. The terminal shows the output of a 'colcon build' command, which is still ongoing. Below the build output, there is a series of log messages from the 'draw_square_action_client' package. These messages show the turtle's position (x, y) and orientation (theta) at various points during the square-drawing process, along with the linear and angular velocities. The final message indicates that the square has been successfully drawn.

```
colcon build [1/1 done] [0 ongoing]
vilma@vilma-Precision-3571: ~/ros2_ws
city=1.0641181468963623, angular_velocity=0.0)
[INFO] [1699790758.938771964] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.00529146194458, y=5.945998668670654, theta=-1.5912097692489624, linear_velo
ity=0.0, angular_velocity=0.007777453400194645)
[INFO] [1699790760.897185634] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.005184650421143, y=5.938431262969971, theta=-1.5849148035049438, linear_velo
city=0.4730067253112793, angular_velocity=0.0)
[INFO] [1699790763.966055997] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.005184650421143, y=5.938431262969971, theta=-1.5849148035049438, linear_velo
city=0.4730067253112793, angular_velocity=0.0)
[INFO] [1699790765.915783308] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.005184650421143, y=5.938431262969971, theta=-1.5849148035049438, linear_velo
city=0.4730067253112793, angular_velocity=0.0)
[INFO] [1699790767.634988977] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=4.998079776763916, y=5.435232162475586, theta=-1.5733232498168945, linear_velo
city=0.0, angular_velocity=0.006579593289643526)
[INFO] [1699790769.224967946] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=4.997697830200195, y=5.262391567230225, theta=-1.5681653022766113, linear_velo
city=0.0, angular_velocity=0.005408311262726784)
[INFO] [1699790770.718561695] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=4.998149394989014, y=5.101295948028564, theta=-1.5679922103881836, linear_velo
city=0.08455748856067657, angular_velocity=0.0)
[INFO] [1699790772.713036915] [draw_square_action_client]: Result: True
[INFO] [1699790772.713188372] [draw_square_action_client]: Finished drawing square
vilma@vilma-Precision-3571: ~/ros2_ws$
```

Thank you!!!
Questions?