

Robot Operating System (ROS2)

Vilma Muço

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

December, 2024



Table of Contents

- Actions
- Exercise
- Homework

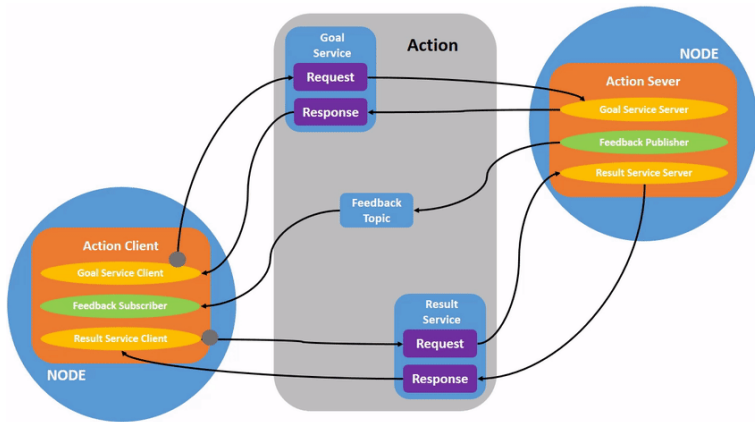


Image taken from docs.ros.org

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
6         self.publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
7
8         velocity_callback_group = MutuallyExclusiveCallbackGroup()
9         self.subscription = self.create_subscription(
10             Pose,
11             "turtle1/pose",
12             self.pose_listener_callback,
13             10,
14             callback_group=velocity_callback_group,
15         )
16
17         action_callback_group = MutuallyExclusiveCallbackGroup()
18         self.action_server = ActionServer(
19             self,
20             GoToPoint,
21             "go_to_point",
22             self.action_callback,
23             callback_group=action_callback_group,
24         )
25
26         self.current_pose = Pose()
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

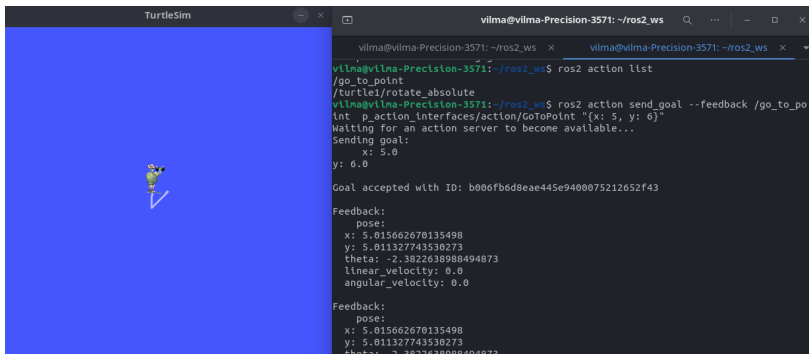
```
1 def main(args=None):
2     rclpy.init(args=args)
3
4     go_to_goal = GoToGoal()
5     executor = MultiThreadedExecutor() # needed to run functions concurrently
6     executor.add_node(go_to_goal)
7
8     executor.spin()
9
10    go_to_goal.destroy_node()
11    rclpy.shutdown()
12
13
14 if __name__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



The screenshot shows a TurtleSim window on the left with a blue background and a small robot icon. On the right is a terminal window titled 'vilma@vilma-Precision-3571: ~/ros2_ws'. The terminal displays the following commands and output:

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 action list
/go_to_point
/turtle1/rotate_absolute
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 action send_goal --feedback /go_to_po
int p_action_interfaces/action/GoToPoint "{x: 5, y: 6}"
Waiting for an action server to become available...
Sending goal:
  x: 5.0
  y: 6.0
Goal accepted with ID: b006fb6d8eae445e9400075212652f43
Feedback:
  pose:
    x: 5.015662670135498
    y: 5.011327743530273
    theta: -2.3822638988494873
    linear_velocity: 0.0
    angular_velocity: 0.0
Feedback:
  pose:
    x: 5.015662670135498
    y: 5.011327743530273
    theta: -2.3822638988494873
```

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()
```

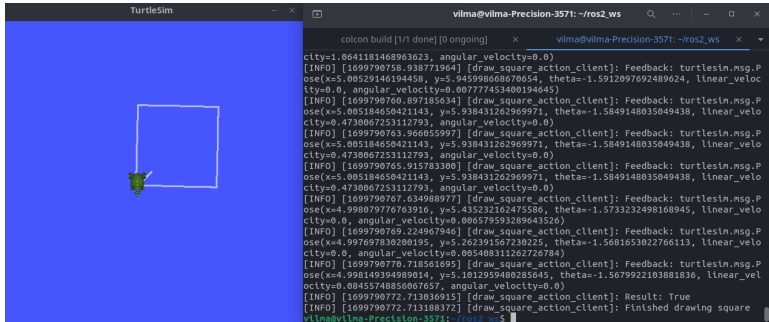
Action Client

```
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



```
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_veloc
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] [1699790765.915783300] [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] [1699790767.034988977] [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=5.1012959480285645, 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$
```

- ▶ Modify the service example code and use the service `/turtle1/set_pen` to draw two circles with 2 different colors
- ▶ Follow the instructions on https://github.com/vilmamuco/bluerov2_exercise



Modify the action example in order to draw four little squares to form a bigger square with twice the length given in the request. Send an email (muco.vilma@gmail.com) with the title [ROS2-COURSE-HOMEWORK-1] containing:

- ▶ a short video/recording of your screen with:
 - ▶ your explanation of the main lines of the code
 - ▶ buiding the package
 - ▶ the turtle creating the big square
- ▶ a zip folder with your code



Thank you!!!
Questions?