Experiment 3: TurtleSim Programming, Publisher, Subscriber, Services, Actions

TurtleSim Programming

Execute in Terminal #1
ros2 run turtlesim turtlesim_node

Execute in Terminal #2
ros2 topic list
ros2 node list
ros2 topic info /turtle1/cmd_vel
ros2 topic info /turtle1/pose
ros2 interface show geometry_msgs/msg/Twist
This expresses velocity in free space broken into its linear and angular parts.
ros2 interface show turtlesim/msg/Pose
Execute in Terminal #3®
ros2 run turtlesim turtle_teleop_key
Execute in Terminal #2
ros2 topic list
ros2 node list
ros2 topic echo /turtle1/cmd_vel
ros2 topic echo/turtle1/pose

ROS2 interfaces:

https://github.com/ros2/example_interfaces
https://github.com/ros2/common_interfaces

You will use 3 nodes:

- The turtlesim_node from the turtlesim package
- A custom node to control the turtle (named “turtle1”) which is already existing in the turtlesim_node. This node can be called turtle_controller.
- A custom node to spawn turtles on the window. This node can be called turtle_spawner.

Execute in Terminal #1
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python turtle_control --dependencies rclpy
colcon build --packages-select turtle_control

Execute in Terminal #2

```
touch turtle_controller.py
chmod +x turtle_controller.py
```

Open src with Visual Studio Application

Enter the code in turtle_controller.py

```
 #!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from my_robot_interface.srv import MoveLocation
import math


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.target_x = 9.0
        self.target_y = 9.0
        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.pose_subscriber_ = self.create_subscription(Pose, "turtle1/pose",
self.callback_turtle_pose, 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)
        self.servce_ = self.create_service(MoveLocation, "move_location",
self.callback_get_distance)
    def callback_turtle_pose(self,msg):
        self.pose_ = msg



    def control_loop(self):
        if self.pose_ == None:
            return
        dist_x = self.target_x - self.pose_.x
        dist_y = self.target_y - self.pose_.y
        distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
        msg = Twist()
        if distance > 0.5:
            msg.linear.x = distance
            goal_theta = math.atan2(dist_y, dist_x)
            diff = goal_theta - self.pose_.theta

            if diff > math.pi:
                diff -= 2*math.pi
            elif diff < -math.pi:
                diff += 2*math.pi
```

```
            msg.angular.z = diff
        else:
            msg.linear.x = 0.0
            msg.angular.z = 0.0

        self.cmd_vel_publisher_.publish(msg)

    def callback_get_distance(self, request, response):
        x = request.loc_x - self.pose_.x
        y = request.loc_y - self.pose_.y

        response.distance = math.sqrt(x * x + y * y)
        return response


def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
        main()
```

Make changes in setup.py

```
from setuptools import find_packages, setup
from glob import glob
import os

package_name = 'turtle_control'

setup(
name=package_name,
version='0.0.0',
packages=find_packages(exclude=['test']),
data_files=[
('share/ament_index/resource_index/packages',
['resource/' + package_name]),
('share/' + package_name, ['package.xml']),
(os.path.join('share', package_name), glob('launch/*')),
],
install_requires=['setuptools'],
zip_safe=True,
maintainer='kashyap',
maintainer_email='kashyap@todo.todo',
```

```
description='TODO: Package description',
license='TODO: License declaration',
tests_require=['pytest'],
entry_points={
'console_scripts': [
"control=turtle_control.p_controller:main"
],
},
)
```

Execute in Terminal #1

```
cd ros2_ws
colcon build –packages-select turtle_control
source install/setup.bash
```

Execute in Terminal #1

```
ros2 run turtlesim turtlesim_node
```

Execute in Terminal #2

```
cd ros2_ws
source install/setup.bash
ros2 run turtle_control control
```

Execute in Terminal #1

```
ros2 service call /move_location my_robot_interface/srv/MoveLocation "{loc_x: 5.0, loc_y: 5.0}"
```

Create a Launch file

```
cd ~/ros2_ws/src/turtle_control
mkdir launch
cd launch
touch turtle.launch.py
chmod +x turtle.launch.py
```

```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
        return LaunchDescription([
        Node(
                package='turtlesim',
                executable='turtlesim_node',
                output='screen'),
        Node(
                package='turtle_control',
                executable='control',
```

```
                output='screen'),
])
```

```
cd ros2_ws
colcon build –packages-select turtle_control
source install/setup.bash
ros2 launch turtle_control turtle.launch.py
```

Exercise2: Create two new files named movement_server.py and movement_client.py.

1. Create a directory named **srv** inside my_robot_interface package
2. Inside this directory, create a file named **MyCustomServiceMessage**.srv

```
string move  # Signal to define movement
             # "Turn right" to make the robot turn in right direction.
             # "Turn left" to make the robot turn in left direction.
             # "Stop" to make the robot stop the movement.
---
bool success
```

3. Modify CMakeLists.txt file
4. Modify package.xml file
5. Compile and source
6. Use in code

```
ros2 interface show my_robot_interface/srv/MyCustomServiceMessage
```

Action Server – Action Client Nodes

Execute in Terminal #1

```
cd ~/ros2_ws/src/my_robot_interface
mkdir action
touch Navigate2D.action

#Goal
int32 secs
---
#Result
string status
---
#Feedback
string feedback
```

package.xml

```
  <depend>rclcpp</depend>
```

```
  <depend>std_msgs</depend>
  <depend>action_msgs</depend>
```

CMakeLists.txt

```
rosidl_generate_interfaces(my_robot_interface
"msg/ManufactureDate.msg"
"srv/SetDate.srv"
"srv/MoveLocation.srv"
"action/Navigate2D.action"
)
```

Execute in Terminal #1

```
colcon build –packages-select my_robot_interface
```

Execute in Terminal #1

```
cd ~/ros2_ws/src/my_package/my_package
touch  action_client.py
chmod +x  action_client.py
```

```
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
from rclpy.executors import MultiThreadedExecutor
from my_robot_interface.action import Navigate2D

class MyActionClient(Node):
    def __init__(self):
        super().__init__('action_client')
        self._action_client = ActionClient(self, Navigate2D, "navigate")

    def send_goal(self, secs):
        goal_msg = Navigate2D.Goal()
        goal_msg.secs = secs
        self._action_client.wait_for_server()
        self._send_goal_future = self._action_client.send_goal_async(goal_msg,
self.feedback_callback)
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected')
            return
        self.get_logger().info('Goal accepted')
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Result: {0}'.format(result.status))
        rclpy.shutdown()
```

```
    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg.feedback
        self.get_logger().info('Received feedback: {0}'.format(feedback.feedback))

def main(args=None):
    rclpy.init(args=args)
    action_client = MyActionClient()
    future = action_client.send_goal(5)
    executor = MultiThreadedExecutor()
    rclpy.spin(action_client, executor=executor)

if __name__ == '__main__':
    main()
```

Execute in Terminal #1

```
cd ~/ros2_ws/src/my_package/my_package
touch  action_server.py
chmod +x  action_server.py
```

Edit the file action_server.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from rclpy.action import ActionServer
import time
from my_robot_interface.action import Navigate2D


class NavigateAction(Node):
    def __init__(self):
        super().__init__("action_server")
        self.action_server_ = ActionServer(
            self, Navigate2D, "navigate", self.navigate_callback)
        self.cmd = Twist()
        self.publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)

    def navigate_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')
        feedback_msg = Navigate2D.Feedback()
        feedback_msg.feedback = "Moving to the left ..."
        for i in range(1, goal_handle.request.secs):
            self.get_logger().info(feedback_msg.feedback)
            goal_handle.publish_feedback(feedback_msg)
            self.cmd.linear.x = 0.3
            self.cmd.angular.z = 0.3
            self.publisher_.publish(self.cmd)
            time.sleep(1)
```

```
        goal_handle.succeed()
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = 0.0
        self.publisher_.publish(self.cmd)
        feedback_msg.feedback = "Finished action server. Robot moved during 5 seconds"
        result = Navigate2D.Result()
        result.status = feedback_msg.feedback
        return result


def main(args=None):
    rclpy.init(args=args)
    node = NavigateAction()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

Edit CmakeLists.txt

```
  entry_points={
        'console_scripts': [
        'sample = my_package.sample:main',
        'robot_publisher = my_package.robot_publisher:main',
        'robot_subscriber = my_package.robot_subscriber:main',
        'add_two_int_server = my_package.add_two_int_server:main',
        'add_two_ints_client = my_package.add_two_ints_client:main',
        'turtlesim_controller = my_package.turtle_controller:main',
        'action_client = my_package.action_client:main',
        'action_server = my_package.action_server:main'
        ],
```

Execute in Terminal #1

```
ros2 run turtlesim turtlesim_node
```

Execute in Terminal #2

```
ros2 run my_package action_client
```

Execute in Terminal #3

```
ros2 run my_package action_server
```