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

Recap:

Setup for launch files:

- Create a new package
- Create a launch/ folder at the root of the package.
- Configure CMakeLists.txt to install files from this launch/ folder.
- Create any number of files you want inside the launch/ folder, ending with .launch.py.

Run a launch file:

- use “colcon build” to install the file.
- source your environment
- launch file with “ros2 launch <package> <name_of_the_file>

First try to design the application by yourself. Don’t write code! Just take a piece of paper and make the design. What nodes should you create? How do the nodes communicate between each other? Which functionality should you add, and where to put them? Etc.

- Directly start on your own (Use the template to start with)
- Work step by step on each functionality/communication.

Client – Server Nodes

Execute in Terminal #1
ros2 interface show example_interfaces/srv/AddTwoInts

Execute in Terminal #1
```
cd ros2_ws/src/my_package/my_package
touch add_two_ints_server.py
chmod +x add_two_ints_server.py
```

Edit add_two_ints_server.py in visual studio editor

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts

class AddTwoIntsServerNode(Node):
    def __init__(self):
```

```
        super().__init__("add_two_ints_server")
        self.server_ = self.create_service(AddTwoInts, "add_two_ints",
self.callback_add_two_ints)
        self.get_logger().info("Add two ints server has been started")

    def callback_add_two_ints(self, request, response):
        response.sum = request.a + request.b
        self.get_logger().info(str(request.a)+ " + " + str(request.b) + " = " + str(response.sum))
        return response

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

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

Add executable name in setup.py

```
 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_ints_server = my_package.add_two_ints_server:main'
        ],
```

Execute in Terminal #1

```
colcon build –packages-select my_package
```

Execute in Terminal #2

```
ros2 run my_package add_two_ints_server
```

Execute in Terminal #3

```
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 3, b: 4}"
```

Ctrl + C in all terminal windows.

Execute in Terminal #1

```
cd ros2_ws/src/my_package/my_package/
touch add_two_ints_client.py
```

chmod +x add_two_ints_client.py

Edit add_two_ints_client.py using visual studio editor

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts
from functools import partial


class AddTwoIntClientNode(Node):
    def __init__(self):
        super().__init__("add_two_ints_client")
        self.call_add_two_int_server(6, 7)

    def call_add_two_int_server(self, a, b):
        client = self.create_client(AddTwoInts, "add_two_ints")
        while not client.wait_for_service(1.0):
            self.get_logger().warn("Waiting for Server Add Two Ints")
        request = AddTwoInts.Request()
        request.a = a
        request.b = b
        future = client.call_async(request)
        future.add_done_callback(
            partial(self.callback_call_two_ints, a=a, b=b))

    def callback_call_two_ints(self, future, a, b):
        try:
            response = future.result()
            self.get_logger().info(str(a) + " + " + str(b) + " = " + str(response.sum))

        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))


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


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

Add executable name in setup.py

```
entry_points={
        'sample = my_package.sample:main',
        'robot_publisher = my_package.robot_publisher:main',
        'robot_subscriber = my_package.robot_subscriber:main',
        'add_two_ints_server = my_package.add_two_ints_server:main',
```

```
        'add_two_ints_client = my_package.add_two_ints_client:main'
        ],
```

Execute in Terminal #1

```
colcon build --packages-select my_package –symlink-install
```

Execute in Terminal #2

```
ros2 run my_package add_two_int_server
```

Execute in Terminal #3

```
ros2 run my_package add_two_ints_client
```

Execute in Terminal #4

```
ros2 node list
```

Execute in Terminal #5

```
ros2 service list
ros2 service type /add_two_ints
ros2 interface show example_interfaces/srv/AddTwoInts
ros2 service call /add_two_int example_interfaces/srv/AddTwoInts
ros2 service call /add_two_int example_interfaces/srv/AddTwoInts "{a: 3, b: 4}"
rqt
plugins→services→service caller
service - /add_two_ints
Enter the values under Expression for a and b
Click call
```

Response is viewed in the second window

Exercise 1: Create a service-client operation to reset the counter value in the number_counter nodes.

The node “number_publisher” publishes a number on the /”number” topic.

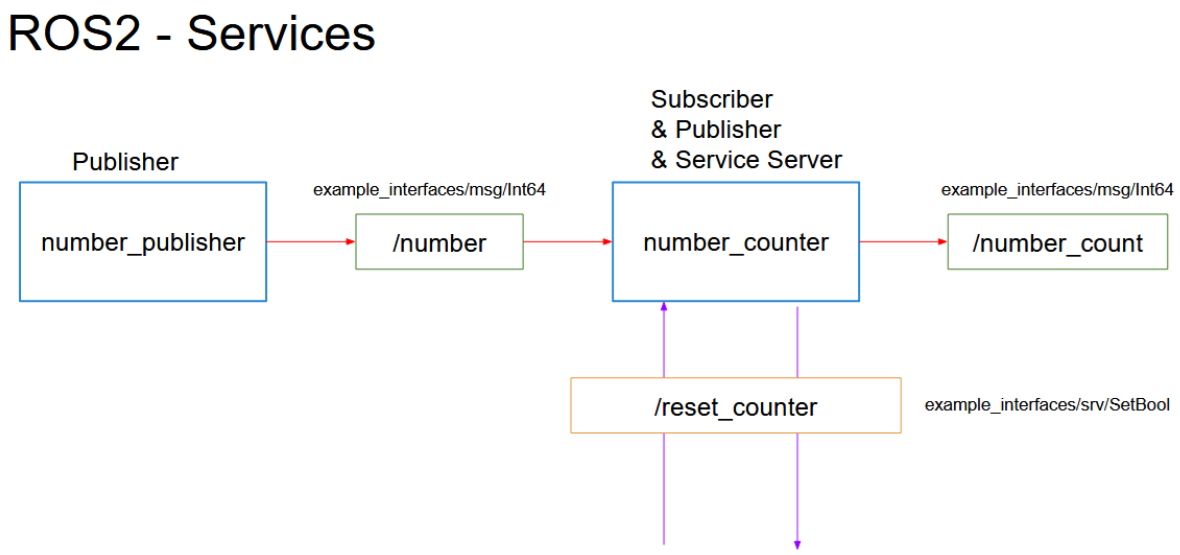
The node “number_counter” gets the number, adds it to a counter, and publishes the counter on the “/number_count” topic.

Add the following ros2 services

Add a functionality to reset the counter to zero:

- Create a service server inside the “number_counter” node.
- Service name: “/reset_counter”
- Service type: example_interfaces/srv/SetBool. Use “ros2 interface show” to discover what’s inside!
- When the server is called, you check the boolean data from the request. If true, you set the counter variable to 0.

We will then call the service directly from the command line. You can also decide - for more practice - to create your own custom node to call this “/reset_counter” service.



Add a functionality to reset the counter to zero:

- Create a service server inside the “number_counter” node.
- Service name: “/reset_counter”
- Service type: example_interfaces/srv/SetBool. Use “ros2 interface show” to discover what’s inside!
- When the server is called, you check the boolean data from the request. If true, you set the counter variable to 0.

We will then call the service directly from the command line. You can also decide - for more practice - to create your own custom node to call this “/reset_counter” service.

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import Int64
from example_interfaces.srv import SetBool

class NumberCounterNode(Node):
    def __init__(self):
        super().__init__("number_counter")
        self.counter_ = 0
        self.number_count_publisher_ = self.create_publisher(Int64, "number_count", 10)
        self.number_subscriber_ = self.create_subscription(Int64, "number",
self.callback_number, 10)
```

```
        self.reset_counter_service_ = self.create_service(SetBool, "reset_counter",
self.callback_reset_counter)
        self.get_logger().info("Node started")


    def callback_number(self, msg):
        self.counter_ += msg.data
        new_msg = Int64()
        new_msg.data = self.counter_
        self.number_count_publisher_.publish(new_msg)
        self.get_logger().info(str(self.counter_))

    def callback_reset_counter(self, request, response):
        if request.data:
            self.counter_ = 0
            response.success = True
            response.message = "Counter is reset"
        else:
            response.success = False
            response.message = "Counter is not reset"
        return response


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

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

Execute in Terminal #1

```
ros2 interface show example_interfaces/srv/SetBool
```

Execute in Terminal #1

```
cd ros2_ws/
colcon build --packages-select my_package
```

Execute in Terminal #1

```
ros2 run my_package number_publisher
```

Execute in Terminal #2

```
ros2 run my_package number_counter
```

Execute in Terminal #3

```
ros2 topic list
ros2 topic echo /number_count
```

Execute in Terminal #4

```
ros2 service call /reset_counter example_interfaces/srv/SetBool "{data: False}"
ros2 service call /reset_counter example_interfaces/srv/SetBool "{data: True}"
```

Custom Services

```
cd ros2_ws/src/my_robot_interface
mkdir srv
cd srv
touch SetDate.srv
```

SetDate.srv

```
string robot_name
int64 date
---
bool success
```

Change CmakeLists.txt as

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

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

Change robot_publisher.py code as

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interface.msg import ManufactureDate
from my_robot_interface.srv import SetDate

class RobotDatePublisher(Node):

    def __init__(self):
        super().__init__("robot_date_publisher")
        self.robot_name_="ROBOT"
        self.publisher_ = self.create_publisher(ManufactureDate,
"robot_manufacturing_date", 10)
        self.timer_ = self.create_timer(0.5, self.publish_news)
        self.set_date_ = self.create_service(SetDate, "set_date", self.callback_set_date)
        self.get_logger().info("Node Started")

    def callback_set_date(self, request, response):
        name = request.robot_name
        date = request.date

        if (name =="ROBOT") and (date==12):
            response.success = True
        else:
```

```
            response.success = False
        return response

    def publish_news(self):
        msg = ManufactureDate()
        msg.date = 12
        msg.month = "March"
        msg.year = 2022
        self.publisher_.publish(msg)

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

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

Execute in Terminal #1

```
colcon build –packages-select my_package
```

Execute in Terminal #1

```
ros2 run my_package robot_publisher
```

Execute in Terminal #2

```
ros2 service list
ros2 service call /set_date my_robot_interface/srv/SetDate "{name: "ROBOT", date: 12}"
```

change robot_subscriber.py code as

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import String, Int32
from my_robot_interface.msg import ManufactureDate
from my_robot_interface.srv import SetDate
from functools import partial

class RobotDateSubscriber(Node):
    def __init__(self):
        super().__init__("robot_date_subscriber")
        self.subscriber_ = self.create_subscription(ManufactureDate,
"robot_manufacturing_date", self.callback_robot_news, 10)
        self.get_logger().info("robot_subscriber Node Started")

    def callback_robot_news(self, msg):
        information ="Manufacturing Date of the ROBOT is " + str(msg.date) + " " +
str(msg.month) + " " + str(msg.year)
        self.get_logger().info(information)
```

```
        self.check_date_server("ROBOT", 12)

    def check_date_server(self, robot_name, date):
        client = self.create_client(SetDate, "set_date")
        while not client.wait_for_service(1.0):
            self.get_logger().warn("Waiting for Server")
        request = SetDate.Request()
        request.robot_name = robot_name
        request.date = date
        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_date_response,
robot_name=robot_name, date=date))

    def callback_date_response(self, future, robot_name, date):
        try:
            response = future.result()
            self.get_logger().info(str(response.success))
        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

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


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

Execute in Terminal #1

```
ros2 run colcon build --packages-select my_package
```

Execute in Terminal #2

```
ros2 run my_package robot_publisher
```

Execute in Terminal #3

```
ros2 run my_package  robot_subscriber
```

TurtleSim Programming

Execute in Terminal #1

```
ros2 run turtlesim turtlesim_node
```

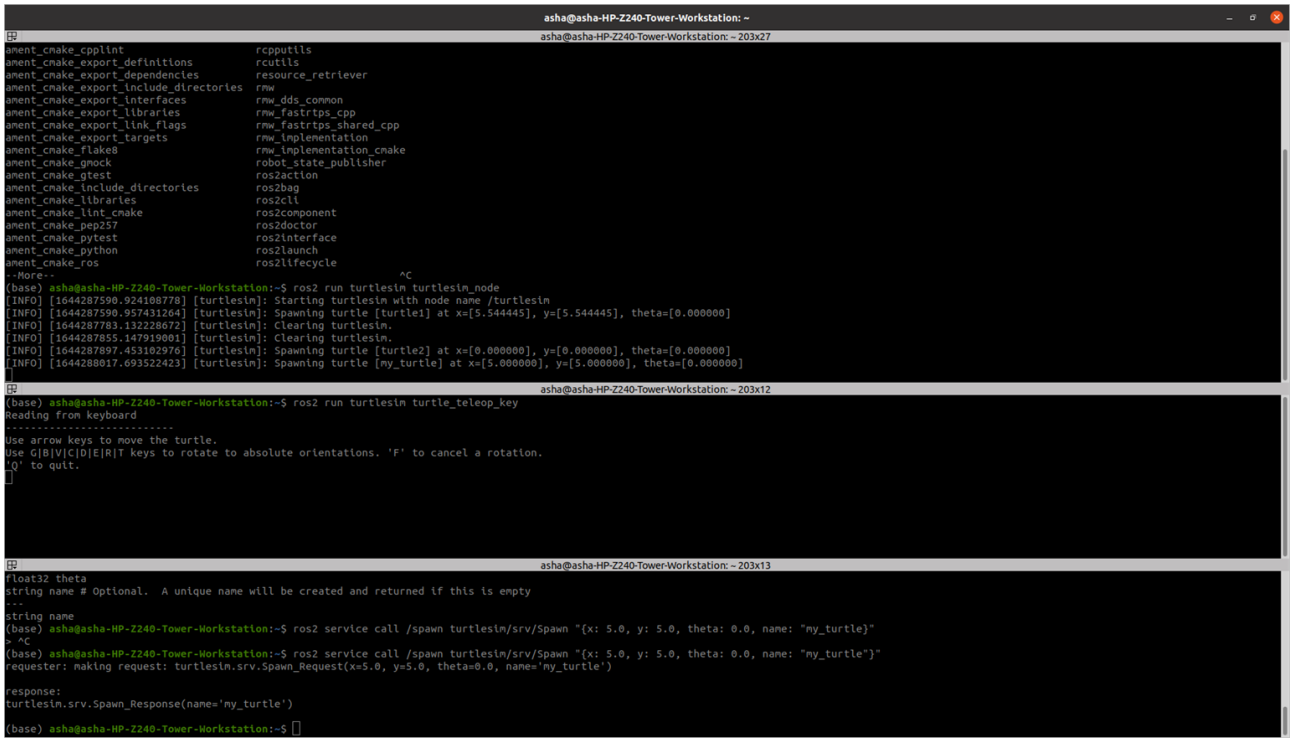
Execute in Terminal #2

```
ros2 run turtlesim turtle_teleop_key
```

Execute in Terminal #3

```
ros2 service list
ros2 service type /clear
ros2 interface show std_srvs/srv/Empty
ros2 service call /clear std_srvs/srv/Empty
ros2 service type /spawn
ros2 interface show turtlesim/srv/Spawn
```

ros2 service call /spawn turtlesim/srv/Spawn
ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.0, y: 5.0, theta: 0.0, name: "my_turtle"}"



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/my_package/my_package

Execute in Terminal #2

touch turtle_controller.py

chmod + 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
import math


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.target_x = 8.0
        self.target_y = 4.0
        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)

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


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

Modify entry_points setup.py as

```
entry_points={
        'console_scripts': [
        'sample = my_package.sample:main',
        'robot_publisher = my_package.robot_publisher:main',
        'robot_subscriber = my_package.robot_subscriber:main',
        'turtlesim_controller = my_package.turtle_controller:main'
        ],
    },
)
```

Modify he package.xml as

```
  <depend>rclpy</depend>
  <depend>example_interfaces</depend>
  <depend>my_robot_interface</depend>
  <depend>turtlesim</depend>
```

Execute in Terminal #1

```
ros2 run turtlesim turtlesim_node
```

Execute in Terminal #2

```
colcon build --packages-select my_package --symlink-install
```

Execute in Terminal #3

ros2 run my_package turtlesim_controller

Execute in Terminal #4

ros2 service list
ros2 service type /spawn
ros2 interface show turtlesim/srv/Spawn

Execute in Terminal #1

cd ros2_ws/my_robot_interface/srv
touch MoveLocation.srv

Edit MoveLocation.srv

```
float32 loc_x
float32 loc_y
---
float32 distance
```

Change CmakeLists.txt as

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

Execute in Terminal #1

cd ~/ros2_ws
colcon build –packages-select my_robot_interface

Send the service request to find the distance between current location and new location.

```
#!/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()
```

Execute in Terminal #1

colcon build –packages-select my_package

Execute in Terminal #2

ros2 run turtlesim turtlesim_node

Execute in Terminal #3

ros2 run my_package turtlesim_controller

Execute in Terminal #1

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

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
```