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.

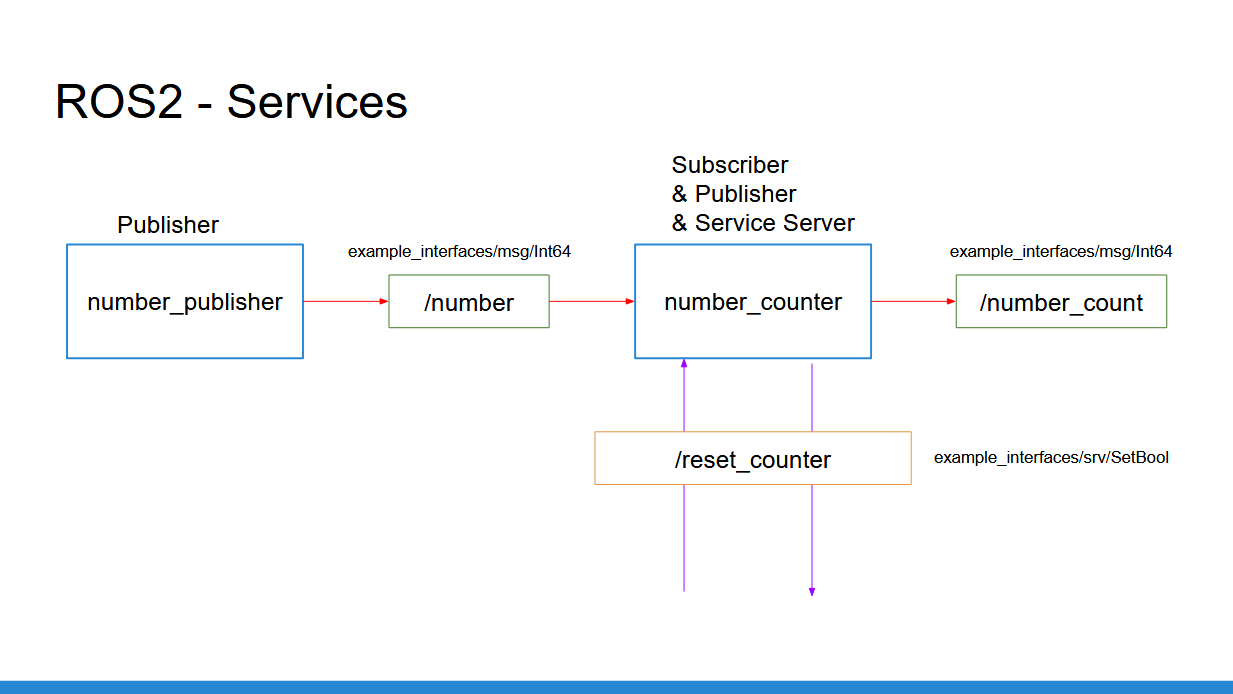
The node “number\_counter” gets the number, adds it to a counter, and publishes the counter on the “/number\_count” topic.

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

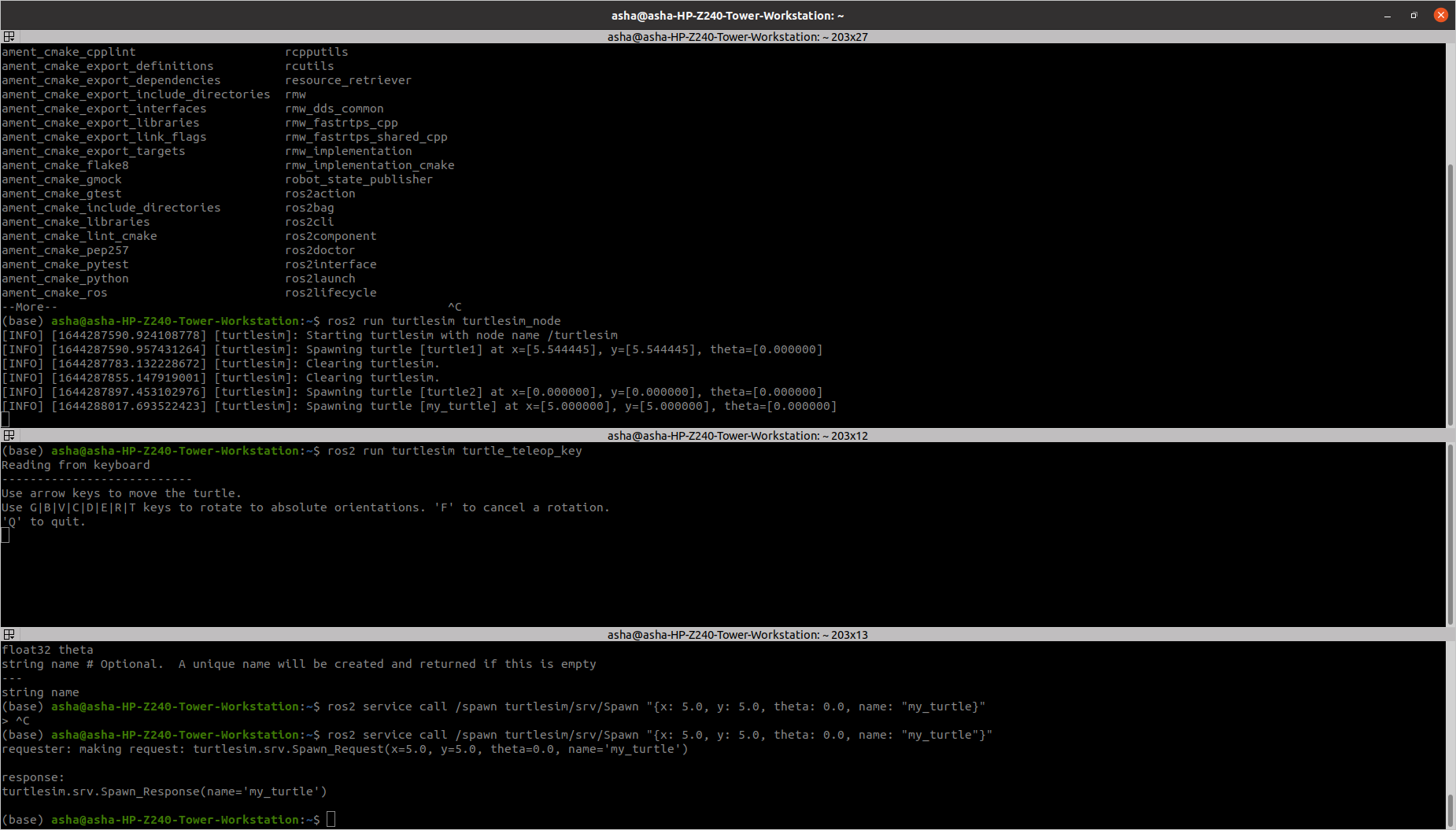
ros2 service call /clear std\_srvs/srv/Empty

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

1. Modify CMakeLists.txt file
2. Modify package.xml file
3. Compile and source
4. 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