MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A Constituent unit of MAHE, Manipal)

ASSIGNMENT 1

Submitted by

Adithya Golwalkar S R

220972012

2nd SEM - M.TECH
DEPARTMENT OF MECHATRONICS

DEPT. OF INDUSTRIAL AUTOMATION AND ROBOTICS MANIPAL INSTITUTE OF TECHNOLOGY, MANIPAL-576104

Exercise 1: Write a launch file pub_sub.launch.py to run the publisher and subscriber node.

CODE SNIPPET

Publisher

pub.py

src > my_package > my_package > pub.py > ...

```python
#!/usr/bin/
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
class Publisher(Node):
    def __init__(self):
        super().__init__('Publisher')
        self.get_logger().info("Publisher Node")
        self.counter_=0
        self.publisher_=self.create_publisher(String,"/message",10)
        self.timer_=self.create_timer(1.0,self.publish_message)

    def publish_message(self):
        msg=String()
        msg.data="Hello"+str(self.counter_)
        self.publisher_.publish(msg)
        self.counter_+=1


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

if __name__=="Main":
    main()

```

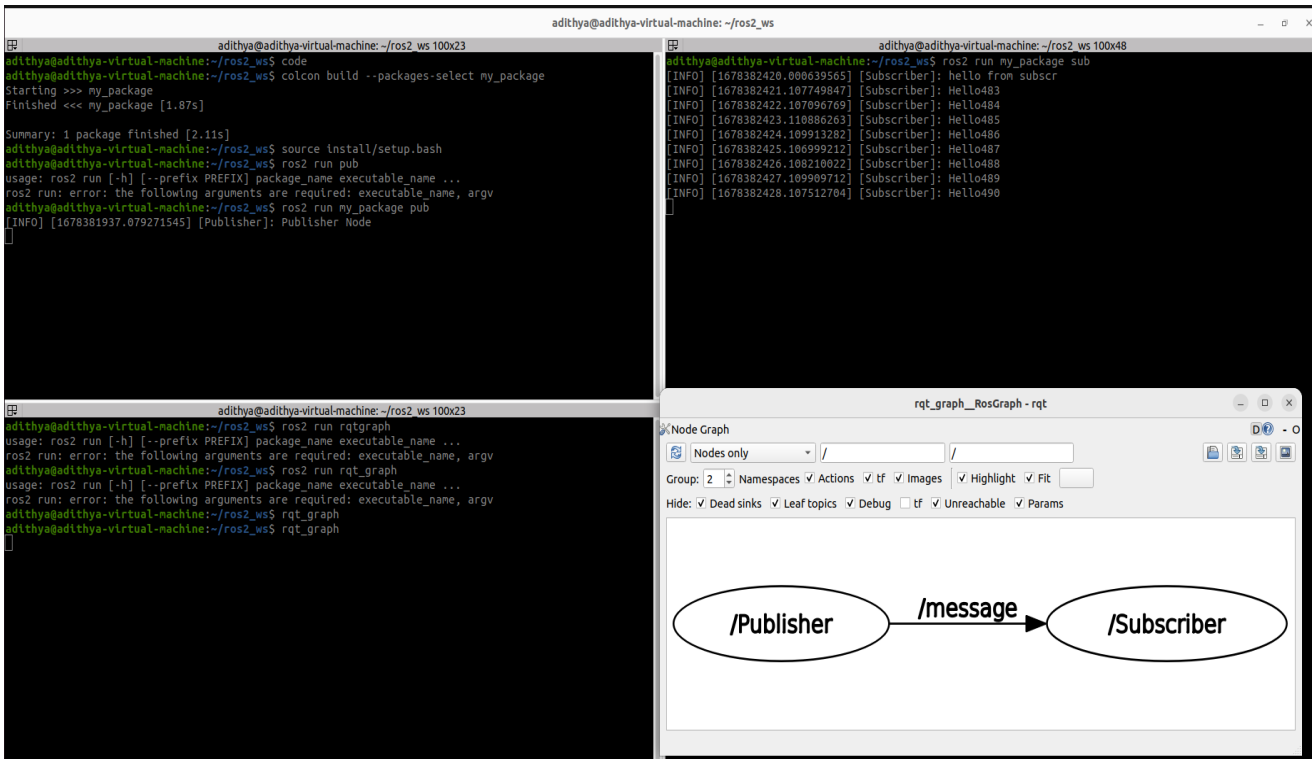
Subscriber

subscriber.py

src > my_package > my_package > subscriber.py > ...

```python
#!usr/bin
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
class subscriber (Node):
    def __init__(self):
        super().__init__("Subscriber")
        self.get_logger().info("hello from subscr")
        self.subscriber=self.create_subscription(String,"/message",self.msg_callback,10)

    def msg_callback(self,msg):
        self.get_logger().info(msg.data)


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


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

RESULT



Exercise 2: Create 2 nodes from scratch. First node has 1 publisher, the second has 1 publisher & 1 subscriber.

CODE SNIPPET

Publisher

1pubs.py

src > my_package > my_package > 1pubs.py > ...

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32

class publisher(Node):
    def __init__(self):
        super().__init__("Publsiher")
        self.get_logger().info("Hello from ROS2, Node is Started")
        # self.counter = 0
        self.publisher_ = self.create_publisher(Int32,"/message",10)
        self.timer_ = self.create_timer(1.0,self.timer_callback)
    def timer_callback(self):
        msg = Int32()
        msg.data = 8    #"Hello " + str(self.counter)
        #self.get_logger().info(msg.data)
        self.publisher_.publish(msg)
        # self.counter+=1

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

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

Subscriber Publisher

2subspubs.py

src > my_package > my_package > 2subspubs.py > ...

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32
from std_msgs.msg import String

class subscriber(Node):
    def __init__(self):
        super().__init__("Subscriber")
        self.get_logger().info("Hello from subscriber...please subscribe to my channel")
        self.counter=0
        self.publisher_ = self.create_publisher(Int32,"/msg_count",10)
        self.subscribe_=self.create_subscription(Int32, "/message",self.subscriber_callback,10)

    def subscriber_callback(self,msg):
        msg_a = Int32()
        msg_int = int(msg.data)
        self.get_logger().info(str(msg.data))
        #msg_a.data = str(msg)+"Hello_Sumukha " + str(self.counter)
        msg_a.data = msg_int + self.counter
        self.publisher_.publish(msg_a)
        self.counter+=1

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


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

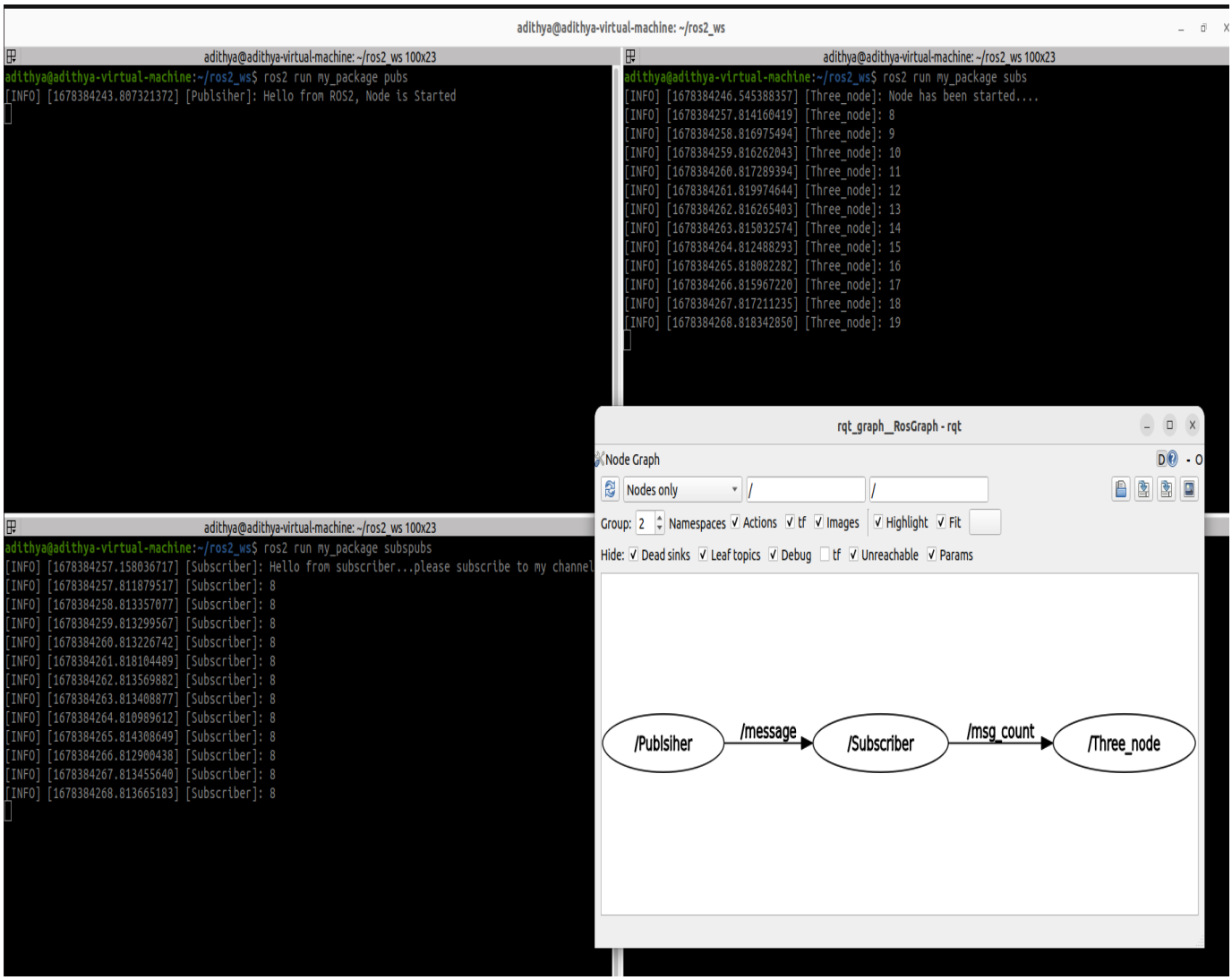
Subscriber

3subs.py

src > my_package > my_package > 3subs.py > three_node > subscriber_callback

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32
import math
import time


class three_node(Node):
    def __init__(self):
        super().__init__("Three_node")
        self.get_logger().info("Node has been started....")
        self.counter = 0
    #    self.timer_ = self.create_timer(1.0,self.timer_callback)
        self.subscribe_=self.create_subscription(Int32, "/msg_count",self.subscriber_callback,10)
        ## called /message topic

    def subscriber_callback(self,msg_a):## Subscribed callback function
        self.get_logger().info(str(msg_a.data))


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


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

RESULT


adithya@adithya-virtual-machine: ~/ros2_ws
adithya@adithya-virtual-machine: ~/ros2_ws 100x23
adithya@adithya-virtual-machine:~/ros2_ws$ ros2 run my_package pubs
[INFO] [1678384243.807321372] [Publsiher]: Hello from ROS2, Node is Started
adithya@adithya-virtual-machine: ~/ros2_ws 100x23
adithya@adithya-virtual-machine:~/ros2_ws$ ros2 run my_package subs
[INFO] [1678384246.545388357] [Three_node]: Node has been started....
[INFO] [1678384257.814160419] [Three_node]: 8
[INFO] [1678384258.816975494] [Three_node]: 9
[INFO] [1678384259.816262043] [Three_node]: 10
[INFO] [1678384260.817289394] [Three_node]: 11
[INFO] [1678384261.819974644] [Three_node]: 12
[INFO] [1678384262.816265403] [Three_node]: 13
[INFO] [1678384263.815032574] [Three_node]: 14
[INFO] [1678384264.812488293] [Three_node]: 15
[INFO] [1678384265.818082282] [Three_node]: 16
[INFO] [1678384266.815967220] [Three_node]: 17
[INFO] [1678384267.817211235] [Three_node]: 18
[INFO] [1678384268.818342850] [Three_node]: 19
adithya@adithya-virtual-machine: ~/ros2_ws 100x23
adithya@adithya-virtual-machine:~/ros2_ws$ ros2 run my_package subspubs
[INFO] [1678384257.158036717] [Subscriber]: Hello from subscriber...please subscribe to my channel
[INFO] [1678384257.811879517] [Subscriber]: 8
[INFO] [1678384258.813357077] [Subscriber]: 8
[INFO] [1678384259.813299567] [Subscriber]: 8
[INFO] [1678384260.813226742] [Subscriber]: 8
[INFO] [1678384261.818104489] [Subscriber]: 8
[INFO] [1678384262.813569882] [Subscriber]: 8
[INFO] [1678384263.813408877] [Subscriber]: 8
[INFO] [1678384264.810989612] [Subscriber]: 8
[INFO] [1678384265.814308649] [Subscriber]: 8
[INFO] [1678384266.812900438] [Subscriber]: 8
[INFO] [1678384267.813455640] [Subscriber]: 8
[INFO] [1678384268.813665183] [Subscriber]: 8
rqt_graph__RosGraph - rqt
Node Graph
Nodes only
Group: 2
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/Publsiher
/message
/Subscriber
/msg_count
/Three_node

Exercise 3: Write a ROS2 program to move the turtle in different shapes. (i) Square (ii) Circle (iii) Triangle (iv)Spiral

Square

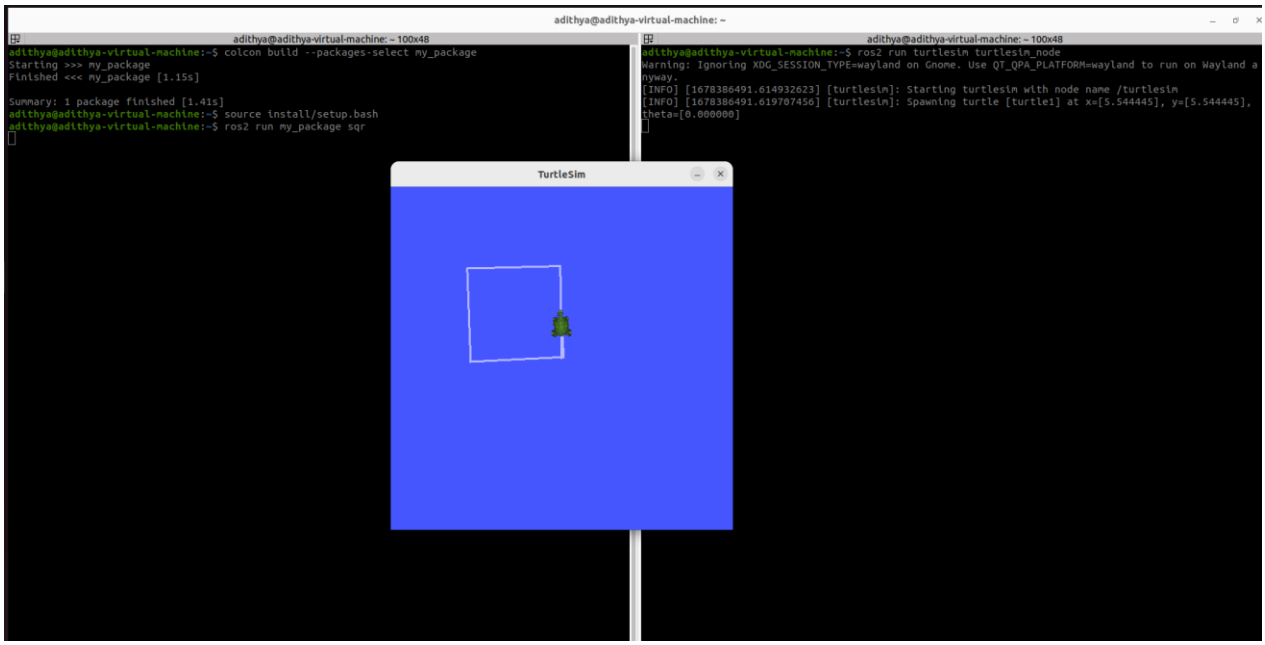
CODE SNIPPET

turtsqr.py

src > my_package > my_package > turtsqr.py > ...

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
import time
# from turtle_control import TurtleControllerNode
class traingle(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.length_y = 2.0
        self.length_x=3.0
        self.theta = math.pi
        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)

    def control_loop(self):
        msg = Twist()
        msg.linear.x = 0.0
        msg.angular.z = (math.pi/2)
        if msg.angular.z == math.pi/2:
            self.cmd_vel_publisher_.publish(msg)
            time.sleep(2)
            msg.angular.z = 0.0
        msg.linear.x = self.length_x
        # msg.linear.y = self.length_y
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(1)
        msg.angular.z = math.pi/2
        msg.linear.x = 0.0
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(2)
        if msg.angular.z == math.pi/2:
            msg.angular.z = 0.0
        msg.linear.x = self.length_x
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(1)
        msg.angular.z = math.pi/2
        msg.linear.x = 0.0
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(2)
        if msg.angular.z == math.pi/2:
            msg.angular.z=0.0
        msg.linear.x = self.length_x
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(2)
def main(args=None):
    rclpy.init(args=args)
    tri = traingle()
    rclpy.spin(tri)
    rclpy.shutdown()

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

RESULT


adithya@adithya-virtual-machine: ~
adithya@adithya-virtual-machine: ~ 100x48
adithya@adithya-virtual-machine:~$ colcon build --packages-select my_package
Starting >>> my_package
Finished <<< my_package [1.15s]
Summary: 1 package finished [1.41s]
adithya@adithya-virtual-machine:~$ source install/setup.bash
adithya@adithya-virtual-machine:~$ ros2 run my_package sqr
TurtleSim
adithya@adithya-virtual-machine: ~ 100x48
adithya@adithya-virtual-machine:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1678386491.614932623] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1678386491.619707456] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

Circle

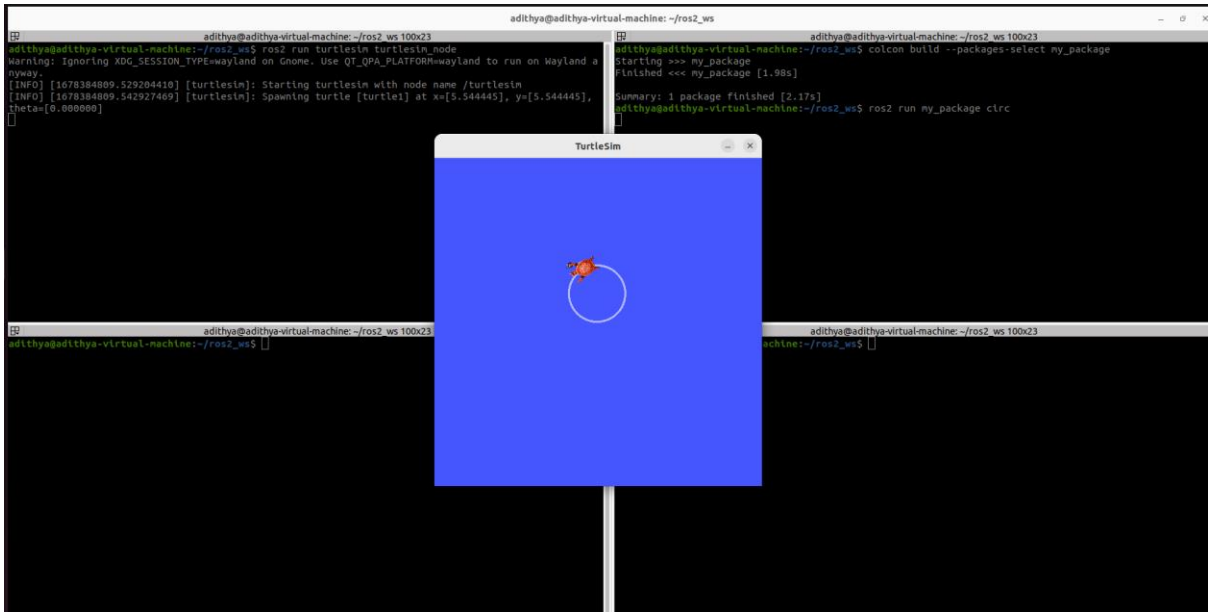
CODE SNIPPET

turtcircle.py

src > my_package > my_package > turtcircle.py > ...

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
# from turtle_control import TurtleControllerNode

class traingle(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.length_y = 3.0
        self.theta = math.pi
        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

      msg = Twist()
      msg.linear.x = self.length_y
      msg.angular.z = self.theta
      self.cmd_vel_publisher_.publish(msg)



def main(args=None):
    rclpy.init(args=args)
    tri = traingle()
    rclpy.spin(tri)
    rclpy.shutdown()

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

RESULT


adithya@adithya-virtual-machine: ~/ros2_ws
adithya@adithya-virtual-machine: ~/ros2_ws 100x23
adithya@adithya-virtual-machine:~/ros2_ws$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland a
nyway.
[INFO] [1670384809.529204410] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1670384809.542927469] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],
theta=[0.000000]
adithya@adithya-virtual-machine: ~/ros2_ws 100x23
adithya@adithya-virtual-machine:~/ros2_ws$ colcon build --packages-select my_package
Starting >>> my_package
Finished <<< my_package [1.98s]
Summary: 1 package finished [2.17s]
adithya@adithya-virtual-machine:~/ros2_ws$ ros2 run my_package circ
TurtleSim
adithya@adithya-virtual-machine: ~/ros2_ws 100x23
adithya@adithya-virtual-machine: ~/ros2_ws 100x23

Triangle

CODE SNIPPET

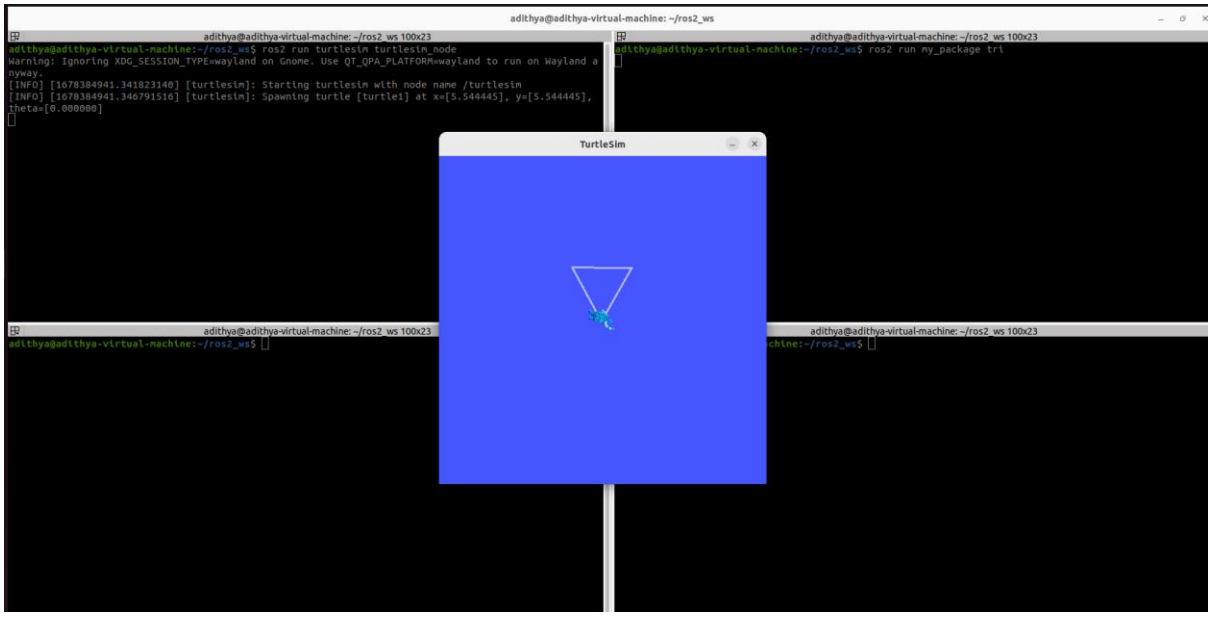
turttria.py

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
import time
# from turtle_control import TurtleControllerNode
class traingle(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.length_y = 2.0
        self.length_x=2.0
        self.theta = math.pi
        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)

    def control_loop(self):
        msg = Twist()
        msg.linear.x = 0.0
        msg.angular.z = (math.pi/3)
        if msg.angular.z == math.pi/3:
            self.cmd_vel_publisher_.publish(msg)
            time.sleep(2)
            msg.angular.z = 0.0
        msg.linear.x = self.length_x
        # msg.linear.y = self.length_y
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(1)
        msg.angular.z = math.pi/1.525
        msg.linear.x = 0.0
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(2)
        if msg.angular.z == math.pi/1.525:
            msg.angular.z = 0.0
        msg.linear.x = self.length_x
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(1)
        msg.angular.z = math.pi/1.525
        msg.linear.x = 0.0
        self.cmd_vel_publisher_.publish(msg)
        time.sleep(2)
```

turttria.py

```python
            msg.linear.x = self.length_x
            self.cmd_vel_publisher_.publish(msg)
            time.sleep(2)
def main(args=None):
    rclpy.init(args=args)
    tri = traingle()
    rclpy.spin(tri)
    rclpy.shutdown()

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

RESULT



Spiral

CODE SNIPPET

turtspiral.py

src > my_package > my_package > turtspiral.py > ...

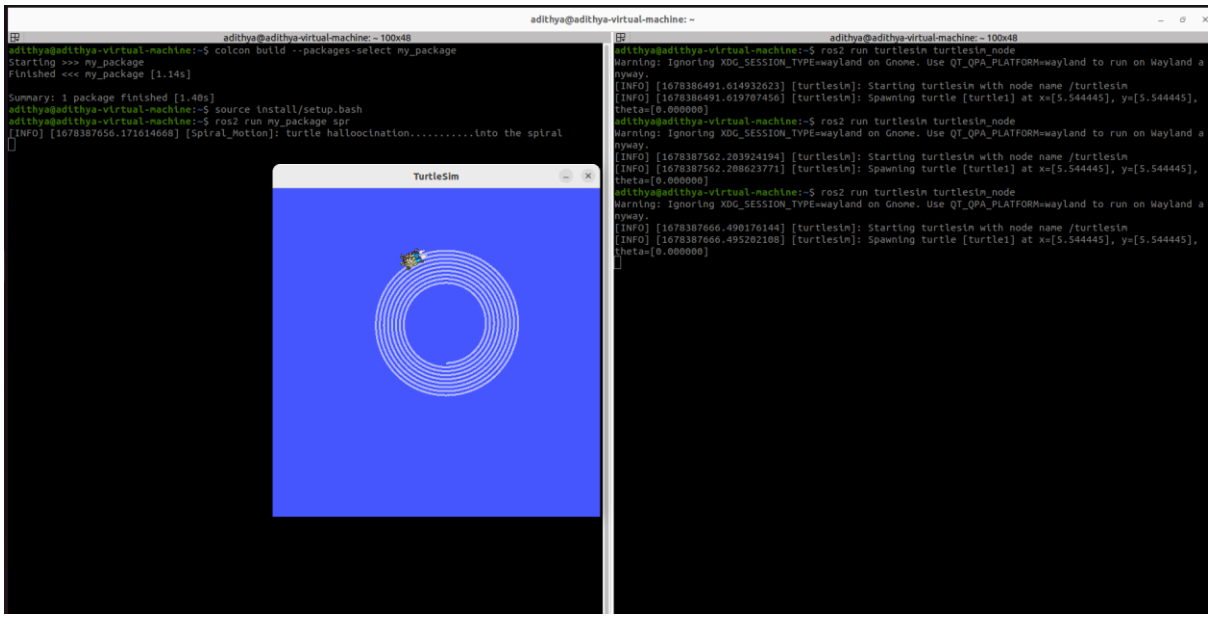
```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from turtlesim.srv import Spawn
from functools import partial
import math
import random
import time

class spiral(Node):
    def __init__(self):
        super().__init__("Spiral_Motion")
        self.velocity_publisher_ = self.create_publisher(Twist,"/turtle1/cmd_vel",10)
        self.pose_subscriber_ = self.create_subscription(Pose,"/turtle/pose",self.pose_callback,10)
        self.pose_x = 0
        self.pose_y = 0
        self.timer_=self.create_timer(1.0, self.publish_velocity)
        self.get_logger().info("turtle halloocination..........into the spiral")
        self.radius = 2.0

    def pose_callback(self,pose_msg=Pose()):
        self.pose_x = pose_msg.x
        self.pose_y = pose_msg.y

        if(round(self.pose_y)==8.0):
            self.get_logger().warn("About to hit wall!")

    def publish_velocity(self):
        vel_msg = Twist()
        vel_msg.linear.x = self.radius

        if self.pose_x < 10.0 and self.pose_y < 10.0:
            vel_msg.angular.z = 22/7
            self.radius += 0.2
        else:
            vel_msg.angular.z = 0.0
            vel_msg.linear.x = 0.0

            self.get_logger().info("done rotation...")
            self.timer_.cancel()
```

turtspiral.py

src > my_package > my_package > turtspiral.py > ...

```python
            self.velocity_publisher_.publish(vel_msg)

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

if __name__ == "_main_":

    main()
```

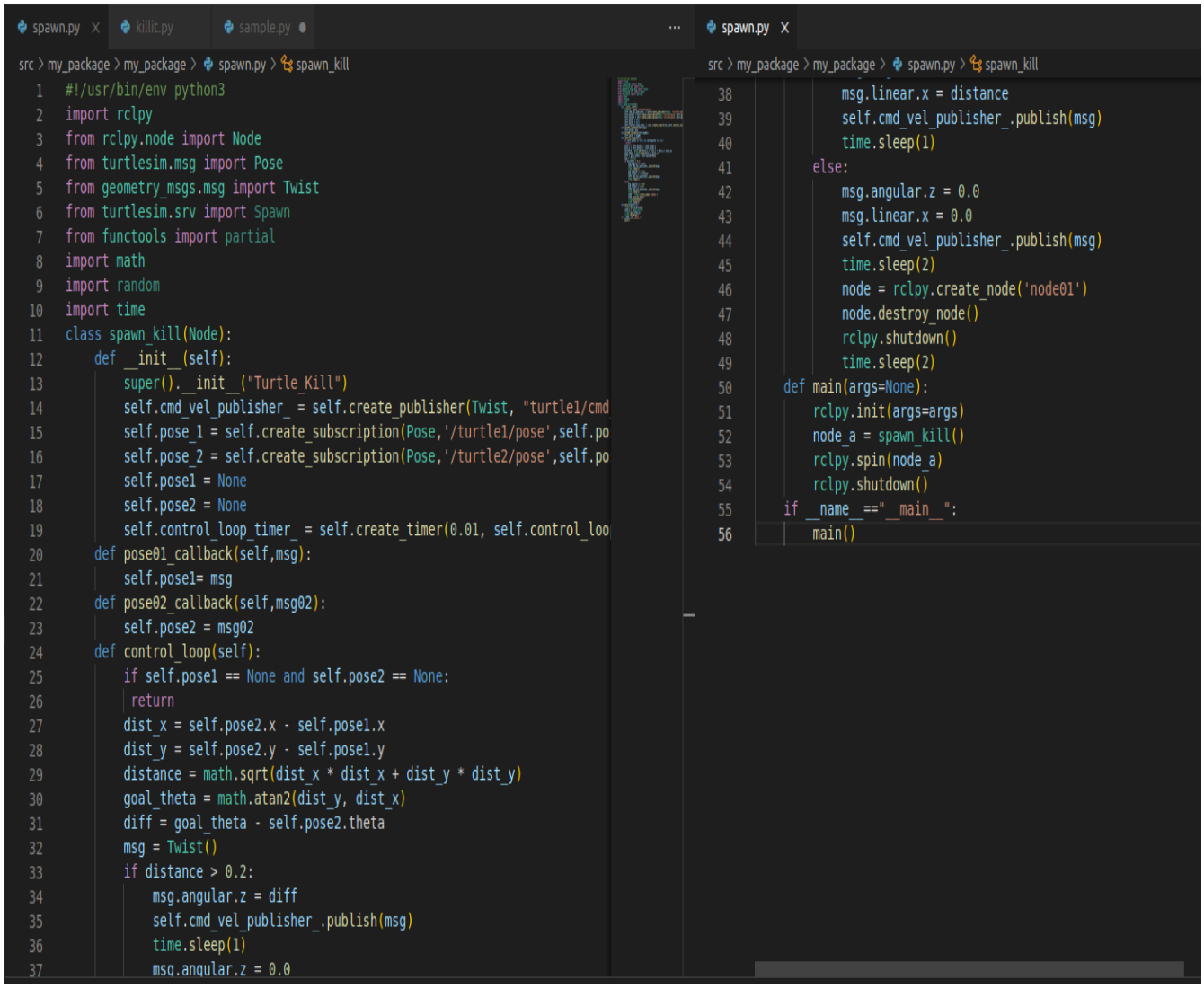
RESULT


adithya@adithya-virtual-machine: ~
adithya@adithya-virtual-machine: ~ 100x48
adithya@adithya-virtual-machine:~$ colcon build --packages-select my_package
Starting >>> my_package
Finished <<< my_package [1.14s]
Summary: 1 package finished [1.40s]
adithya@adithya-virtual-machine:~$ source install/setup.bash
adithya@adithya-virtual-machine:~$ ros2 run my_package spr
[INFO] [1678387656.171614668] [Spiral_Motion]: turtle halloocination..........into the spiral
TurtleSim
adithya@adithya-virtual-machine: ~ 100x48
adithya@adithya-virtual-machine:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1678386491.614932623] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1678386491.619707456] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
adithya@adithya-virtual-machine:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1678387562.203924194] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1678387562.208623771] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
adithya@adithya-virtual-machine:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1678387666.490176144] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1678387666.495202108] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

Exercise 4: Write a ROS2 code to spawn a new turtle (default: turtle2) in a random location. Modify the code to move the turtle1 to turtle2 location and kill it. Use PID algorithm to navigate the turtle.

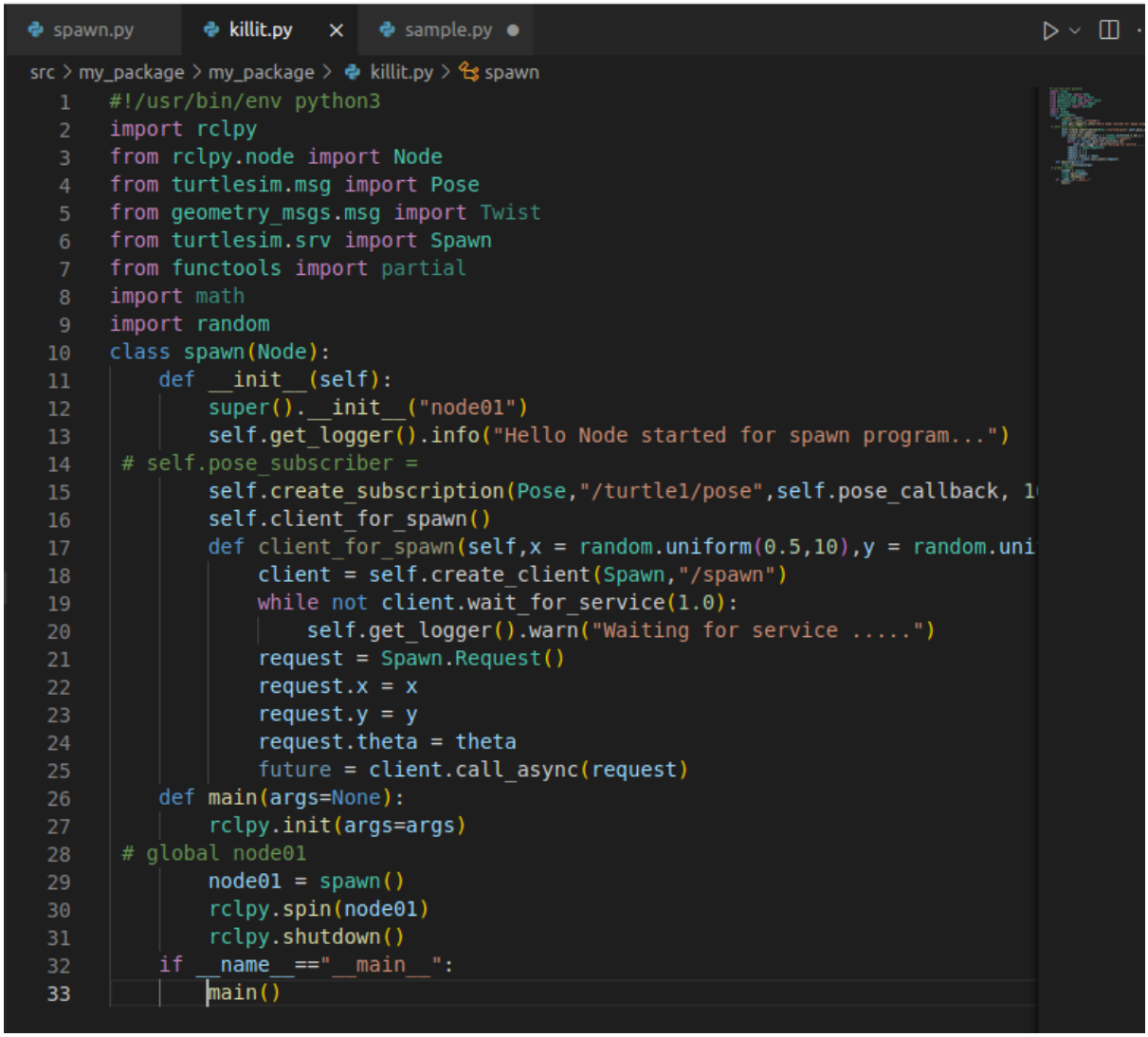
CODE SNIPPET

Spawning



```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from turtlesim.srv import Spawn
from functools import partial
import math
import random
import time
class spawn_kill(Node):
    def __init__(self):
        super().__init__("Turtle_Kill")
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd
        self.pose_1 = self.create_subscription(Pose,'/turtle1/pose',self.po
        self.pose_2 = self.create_subscription(Pose,'/turtle2/pose',self.po
        self.pose1 = None
        self.pose2 = None
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loo
    def pose01_callback(self,msg):
        self.pose1= msg
    def pose02_callback(self,msg02):
        self.pose2 = msg02
    def control_loop(self):
        if self.pose1 == None and self.pose2 == None:
            return
        dist_x = self.pose2.x - self.pose1.x
        dist_y = self.pose2.y - self.pose1.y
        distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
        goal_theta = math.atan2(dist_y, dist_x)
        diff = goal_theta - self.pose2.theta
        msg = Twist()
        if distance > 0.2:
            msg.angular.z = diff
            self.cmd_vel_publisher_.publish(msg)
            time.sleep(1)
            msg.angular.z = 0.0
            msg.linear.x = distance
            self.cmd_vel_publisher_.publish(msg)
            time.sleep(1)
        else:
            msg.angular.z = 0.0
            msg.linear.x = 0.0
            self.cmd_vel_publisher_.publish(msg)
            time.sleep(2)
            node = rclpy.create_node('node01')
            node.destroy_node()
            rclpy.shutdown()
            time.sleep(2)
    def main(args=None):
        rclpy.init(args=args)
        node_a = spawn_kill()
        rclpy.spin(node_a)
        rclpy.shutdown()
    if __name__=="__main__":
        main()
```

Killing


spawn.py
killit.py
sample.py
src > my_package > my_package > killit.py > spawn
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from turtlesim.srv import Spawn
from functools import partial
import math
import random
class spawn(Node):
def __init__(self):
super().__init__("node01")
self.get_logger().info("Hello Node started for spawn program...")
# self.pose_subscriber =
self.create_subscription(Pose,"/turtle1/pose",self.pose_callback, 1
self.client_for_spawn()
def client_for_spawn(self,x = random.uniform(0.5,10),y = random.uni
client = self.create_client(Spawn,"/spawn")
while not client.wait_for_service(1.0):
self.get_logger().warn("Waiting for service .....")
request = Spawn.Request()
request.x = x
request.y = y
request.theta = theta
future = client.call_async(request)
def main(args=None):
rclpy.init(args=args)
# global node01
node01 = spawn()
rclpy.spin(node01)
rclpy.shutdown()
if __name__=="__main__":
main()

RESULT