28.ROS2 coordinate transformation TF2 case

1. Case introduction

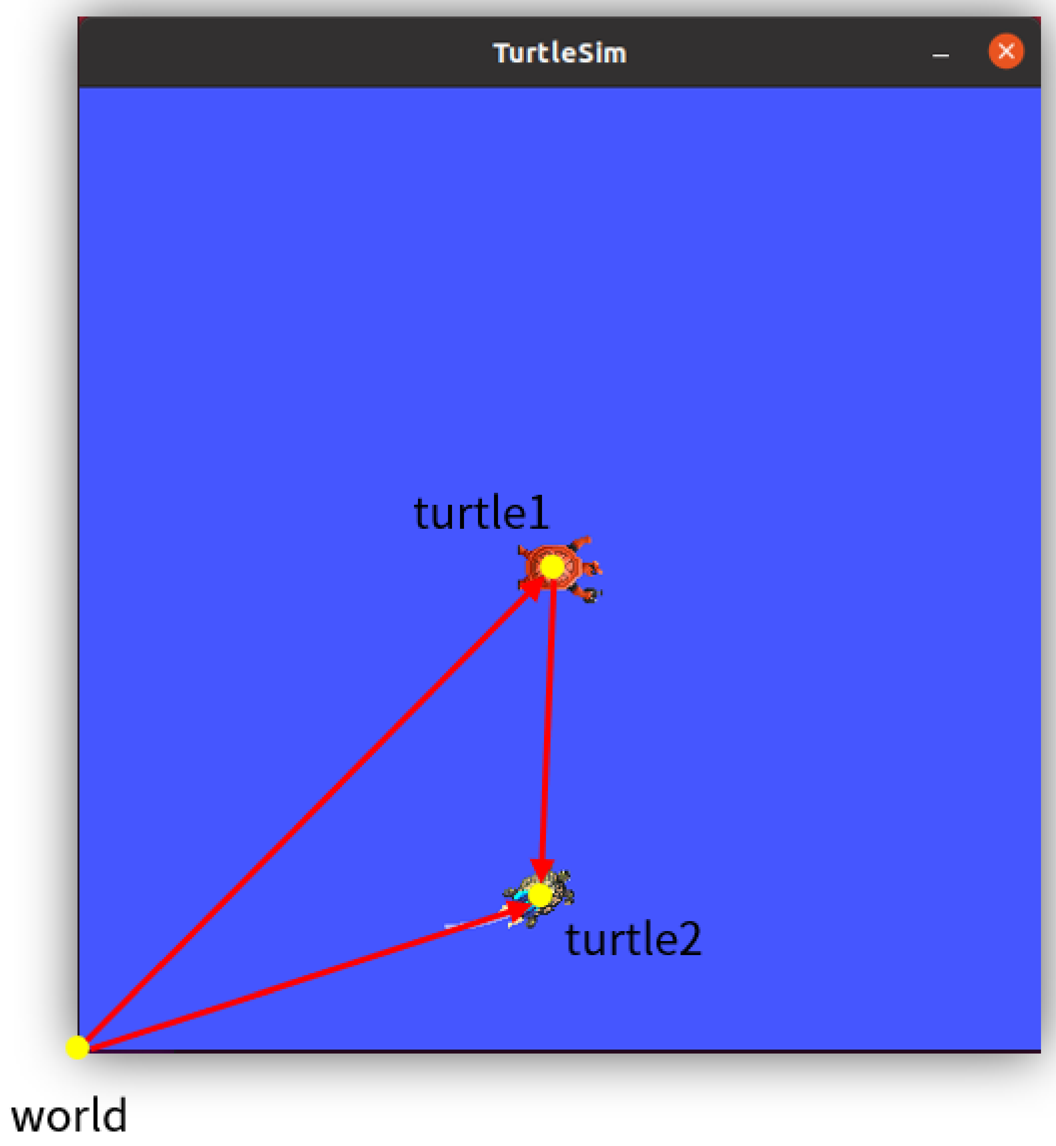
In the previous lesson, we explained the TF relationship in the turtle following case provided by the system. In this lesson, we will implement this function ourselves.

2. Source code path

For the complete code, please view the following path in docker:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/pkg_tf
```

3. Principle analysis



In the simulator of two turtles, we can define three coordinate systems. For example, the global reference system of the simulator is called world. The turtle1 and turtle2 coordinate systems are at the center points of the two turtles. In this way, the relative positions of the turtle1 and world coordinate systems , it can represent the position of turtle 1, and the same is true for turtle 2.

To realize the movement of turtle 2 to turtle 1, we make a connection between the two and add an arrow. How about it? Do you remember the vector calculation you learned in high school? We say that the description method of coordinate transformation is vector, so in this following routine, TF can be used to solve it well.

The length of the vector represents the distance, and the direction represents the angle. With the distance and angle, we can calculate the speed by setting a random time. Then the speed topic is encapsulated and released, and the Turtle 2 can start moving.

Therefore, the core of this routine is to calculate the vector through the coordinate system. The two turtles will continue to move, and the vector must also be calculated according to a certain period, which requires the use of TF's dynamic broadcast and monitoring.

4. Create a new function package

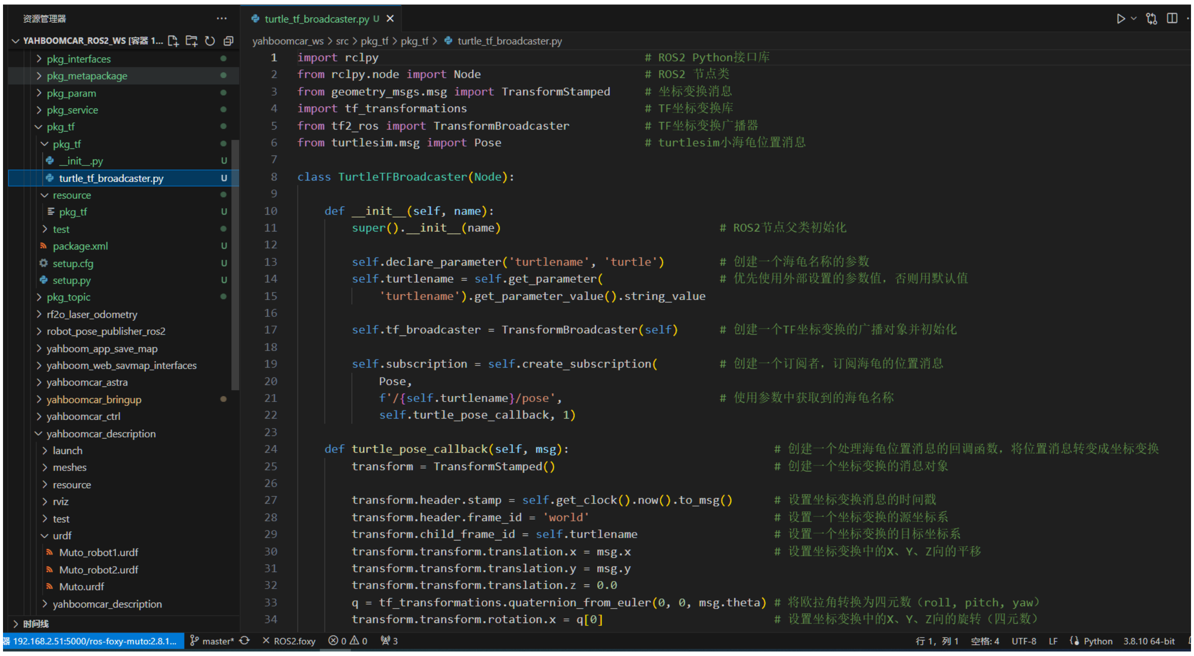
1. Execute the following command in docker:

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws/src
ros2 pkg create pkg_tf --build-type ament_python --dependencies rclpy --node-
name turtle_tf_broadcaster
```

After executing the above command, the pkg_tf function package will be created, and a turtle_tf_broadcaster node will be created, and the relevant configuration files have been configured. Add the following code to the [turtle_tf_broadcaster.py] file:

```
import rclpy                                      # ROS2 Python interface
library
from rclpy.node import Node                       # ROS2 Node class
from geometry_msgs.msg import TransformStamped    # Coordinate Transformation
Message
import tf_transformations                         # TF coordinate
transformation library
from tf2_ros import TransformBroadcaster          # TF coordinate
transformation broadcaster
from turtlesim.msg import Pose                    # turtlesim baby turtle
location message

class TurtleTFBroadcaster(Node):

    def __init__(self, name):
        super().__init__(name)                                # ROS2 node parent
class initialization

        self.declare_parameter('turtlename', 'turtle')        # Create a turtle
name parameter
        self.turtlename = self.get_parameter(                 # Priority is
given to using externally set parameter values, otherwise the default values are
used.
            'turtlename').get_parameter_value().string_value

        self.tf_broadcaster = TransformBroadcaster(self)      # Create a
broadcast object of TF coordinate transformation and initialize it

        self.subscription = self.create_subscription(         # Create a
subscriber to subscribe to the turtle's location information
            Pose,
            f'/{self.turtlename}/pose',                       # Use the turtle
name obtained in the parameters
            self.turtle_pose_callback, 1)

    def turtle_pose_callback(self, msg):                              # Create a
callback function that handles turtle position messages and converts the position
messages into coordinate transformations
        transform = TransformStamped()                                # Create a
coordinate transformation message object

        transform.header.stamp = self.get_clock().now().to_msg()      # Set the
timestamp of the coordinate transformation message
        transform.header.frame_id = 'world'                           # Set a
source coordinate system for coordinate transformation
        transform.child_frame_id = self.turtlename                    # Set a
target coordinate system for coordinate transformation
```

```
        transform.transform.translation.x = msg.x                     # Set the
translation in the X, Y, and Z directions in coordinate transformation
        transform.transform.translation.y = msg.y
        transform.transform.translation.z = 0.0
        q = tf_transformations.quaternion_from_euler(0, 0, msg.theta) # Convert
Euler angles to quaternions (roll, pitch, yaw)
        transform.transform.rotation.x = q[0]                         # Set the
rotation in the X, Y, and Z directions in coordinate transformation (quaternion)
        transform.transform.rotation.y = q[1]
        transform.transform.rotation.z = q[2]
        transform.transform.rotation.w = q[3]

        # Send the transformation
        self.tf_broadcaster.sendTransform(transform)     # Broadcast coordinate
transformation. After the turtle position changes, the coordinate transformation
information will be updated in time.

def main(args=None):
    rclpy.init(args=args)                                # ROS2 Python interface
initialization
    node = TurtleTFBroadcaster("turtle_tf_broadcaster")  # Create a ROS2 node
object and initialize it
    rclpy.spin(node)                                     # Loop waiting for ROS2
to exit
    node.destroy_node()                                  # Destroy node object
    rclpy.shutdown()                                     # Close the ROS2 Python
interface
```



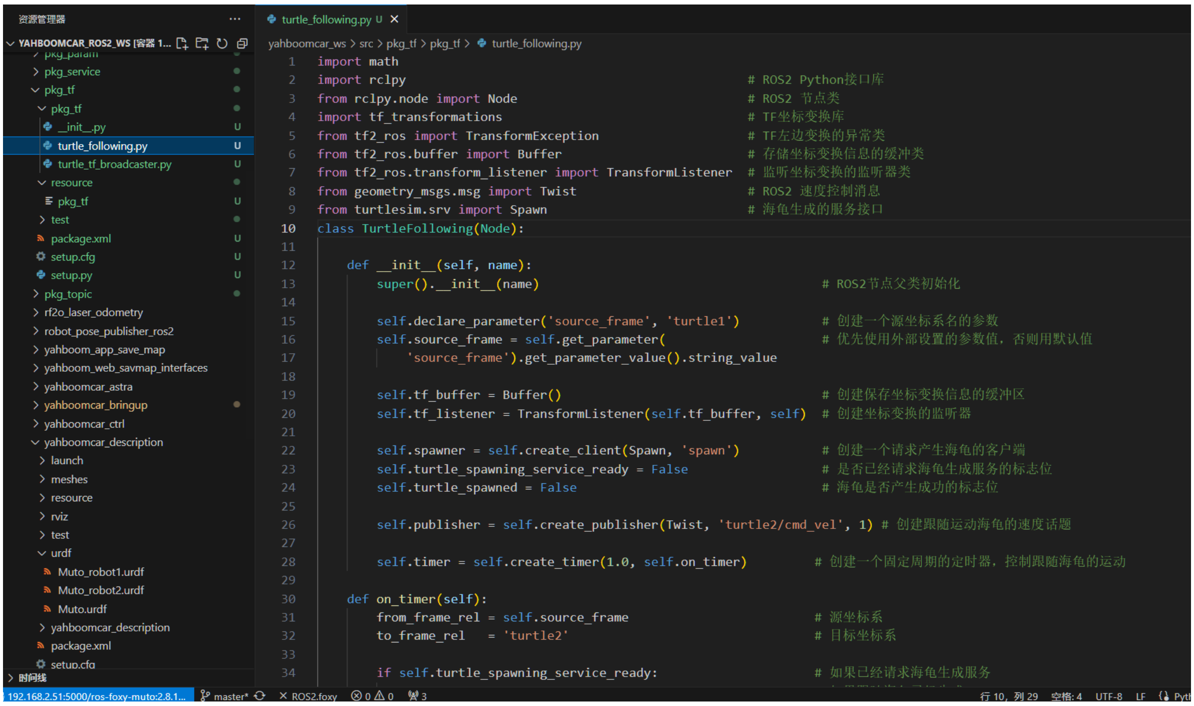
2. Next, create a new [turtle_following.py] file in the same directory as turtle_tf_broadcaster.py and add the following code:

```
import math
import rclpy                                      # ROS2 Python
interface library
from rclpy.node import Node                       # ROS2 Node class
import tf_transformations                         # TF coordinate
transformation library
```

```
from tf2_ros import TransformException                     # The exception class
of the left transformation of TF
from tf2_ros.buffer import Buffer                          # Buffer class that
stores coordinate transformation information
from tf2_ros.transform_listener import TransformListener  # Listener class for
monitoring coordinate transformations
from geometry_msgs.msg import Twist                        # ROS2 speed control
messages
from turtlesim.srv import Spawn                            # Service interface
generated by turtle
class TurtleFollowing(Node):

    def __init__(self, name):
        super().__init__(name)                                       # ROS2 node
parent class initialization

        self.declare_parameter('source_frame', 'turtle1')           # Create a
parameter with the name of the source coordinate system
        self.source_frame = self.get_parameter(                      # Priority
is given to using externally set parameter values, otherwise the default values
are used.
            'source_frame').get_parameter_value().string_value

        self.tf_buffer = Buffer()                                    # Create a
buffer that holds coordinate transformation information
        self.tf_listener = TransformListener(self.tf_buffer, self)  # Create a
coordinate transformation listener

        self.spawner = self.create_client(Spawn, 'spawn')           # Create a
client that requests spawned turtles
        self.turtle_spawning_service_ready = False                   # Whether
the turtle generation service flag has been requested
        self.turtle_spawned = False                                  # Whether
the turtle generates a success flag

        self.publisher = self.create_publisher(Twist, 'turtle2/cmd_vel', 1) #
Create a speed topic that follows a moving turtle

        self.timer = self.create_timer(1.0, self.on_timer)         # Create a
timer with a fixed period to control the movement of the turtle

    def on_timer(self):
        from_frame_rel = self.source_frame                          # Source
coordinate system
        to_frame_rel   = 'turtle2'                                  # target
coordinate system

        if self.turtle_spawning_service_ready:                      # If the
turtle spawning service has been requested
            if self.turtle_spawned:                                 # If
following turtle has spawned
                try:
                    now = rclpy.time.Time()                         # Get the
current time of the ROS system
                    trans = self.tf_buffer.lookup_transform(        # Monitor the
coordinate transformation from the source coordinate system to the target
coordinate system at the current moment
                        to_frame_rel,
```

```
                            from_frame_rel,
                            now)
                except TransformException as ex:                    # If
coordinate transformation acquisition fails, an exception report will be
entered.
                    self.get_logger().info(
                        f'Could not transform {to_frame_rel} to
{from_frame_rel}: {ex}')
                    return

                msg = Twist()                                       # Create
speed control message
                scale_rotation_rate = 1.0                           # Calculate
the angular velocity based on the turtle's angle
                msg.angular.z = scale_rotation_rate * math.atan2(
                    trans.transform.translation.y,
                    trans.transform.translation.x)

                scale_forward_speed = 0.5                           # Calculate
linear velocity based on turtle distance
                msg.linear.x = scale_forward_speed * math.sqrt(
                    trans.transform.translation.x ** 2 +
                    trans.transform.translation.y ** 2)

                self.publisher.publish(msg)                         # Issue a
speed command and the turtle follows the movement
            else:                                                   # If
following turtle is not spawned
                if self.result.done():                              # Check
whether turtles are spawned
                    self.get_logger().info(
                        f'Successfully spawned {self.result.result().name}')
                    self.turtle_spawned = True
                else:                                               # Still no
following turtle spawned
                    self.get_logger().info('Spawn is not finished')
        else:                                                       # If the
turtle spawning service is not requested
            if self.spawner.service_is_ready():                     # If the
turtle spawn server is ready
                request = Spawn.Request()                           # Create a
requested data
                request.name = 'turtle2'                            # Set the
content of the requested data, including turtle name, xy position, and attitude
                request.x = float(4)
                request.y = float(2)
                request.theta = float(0)

                self.result = self.spawner.call_async(request)      # Send a
service request
                self.turtle_spawning_service_ready = True           # Set the
flag bit to indicate that the request has been sent
            else:
                self.get_logger().info('Service is not ready')      # Turtle
spawn server is not ready yet


def main(args=None):
```

```
    rclpy.init(args=args)                        # ROS2 Python interface
initialization
    node = TurtleFollowing("turtle_following")  # Create a ROS2 node object and
initialize it
    rclpy.spin(node)                             # Loop waiting for ROS2 to exit
    node.destroy_node()                          # Destroy node object
    rclpy.shutdown()                             # Close the ROS2 Python
interface
```



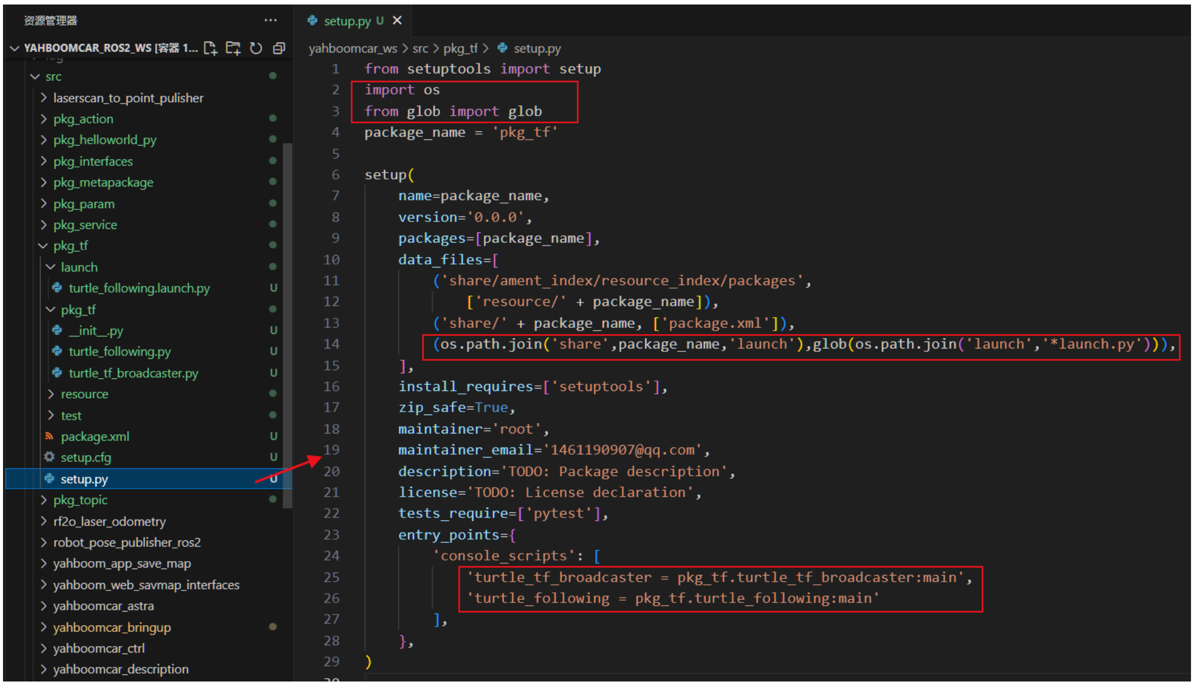
3. Create a new launch folder under the pkg_tf function package, create a new [turtle_following.launch.py] file in the launch folder, and add the following content:

```
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def generate_launch_description():

    return LaunchDescription([
        DeclareLaunchArgument('target_frame', default_value='turtle1',
description='Target frame name.'),
        Node(
            package='turtlesim',
            executable='turtlesim_node',
        ),
        Node(
            package='pkg_tf',
            executable='turtle_tf_broadcaster',
            name='broadcaster1',
            parameters=[
                {'turtlename': 'turtle1'}
            ]
        ),
        Node(
```

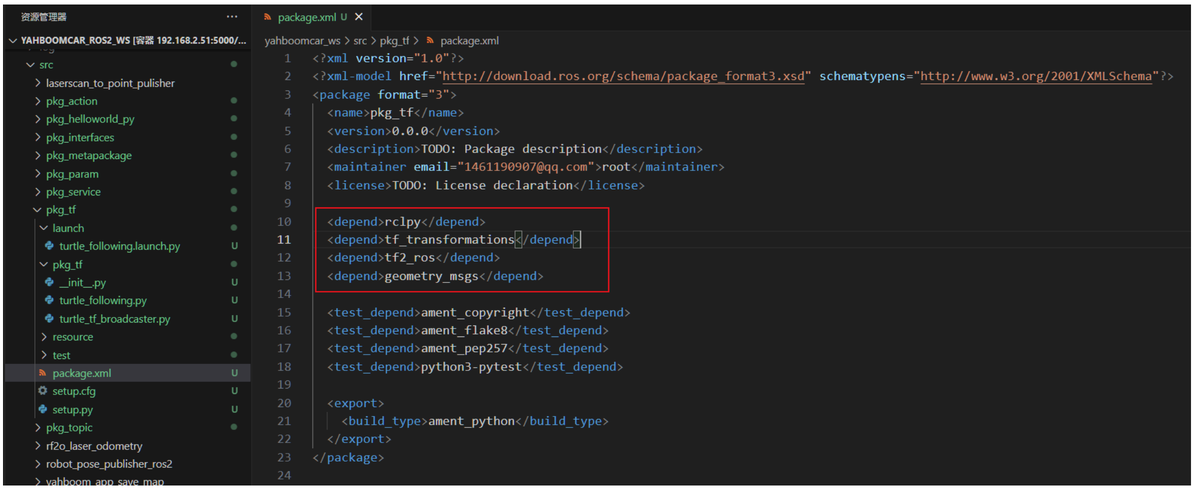
```
            package='pkg_tf',
            executable='turtle_tf_broadcaster',
            name='broadcaster2',
            parameters=[
                {'turtlename': 'turtle2'}
            ]
        ),
        Node(
            package='pkg_tf',
            executable='turtle_following',
            name='listener',
            parameters=[
                {'target_frame': LaunchConfiguration('target_frame')}
            ]
        ),
    ])
```

5. Edit configuration file

5.1. Configuration in setup.py



5.2. Configuration in package.xml



6. Compile workspace

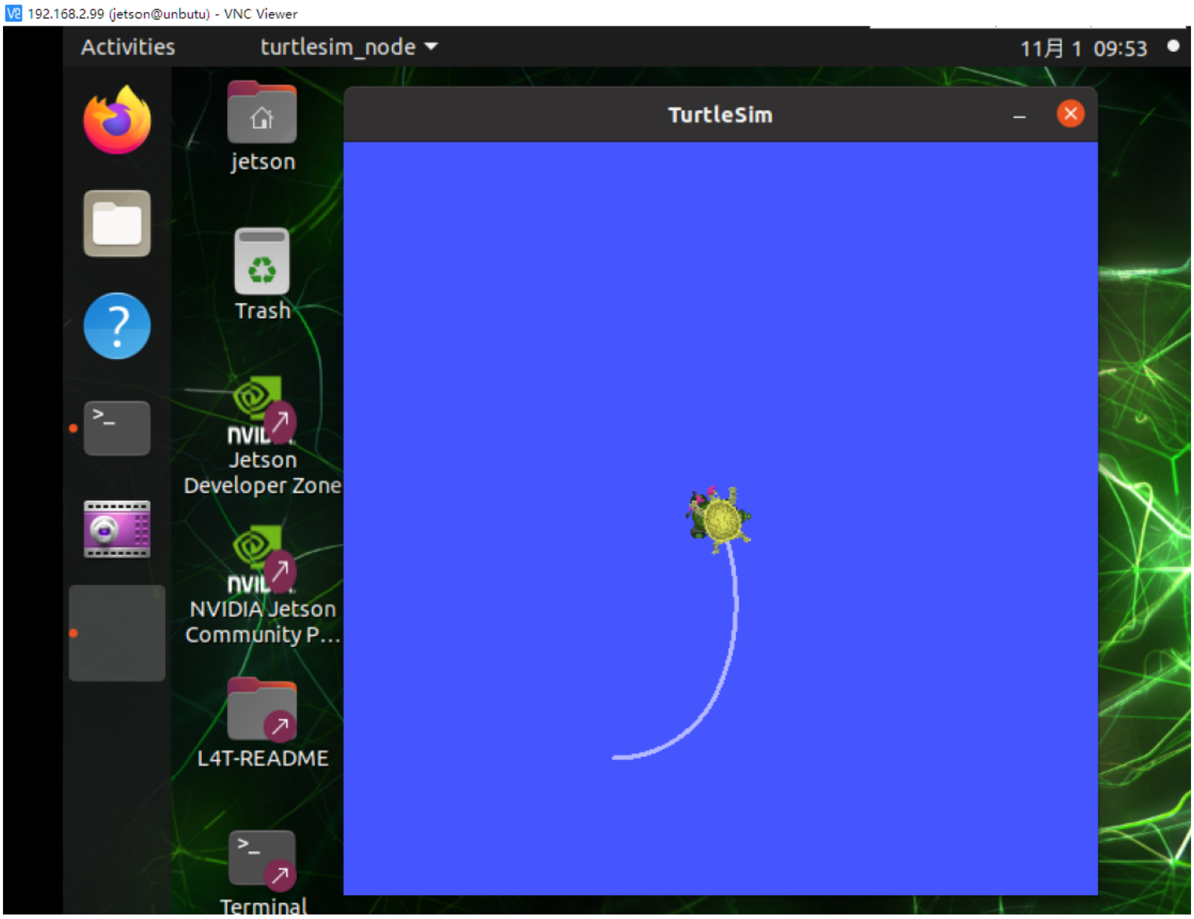
```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_tf
source install/setup.bash
```

7. Run the program

Make sure that docker has enabled GUI display, then open a terminal in docker and execute:

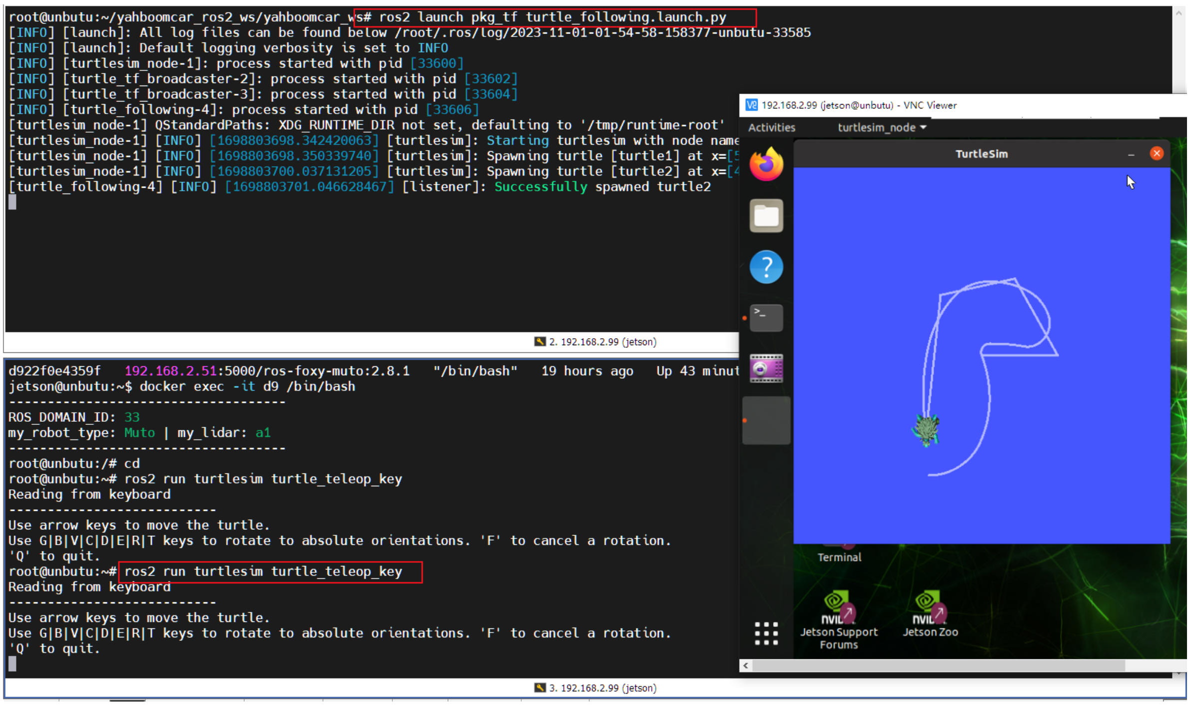
```
ros2 launch pkg_tf turtle_following.launch.py
```

After running, you can see on muto's host vnc: two small turtles are generated, and one of them moves closer to the other.



Open another terminal in docker and execute:

```
ros2 run turtlesim turtle_teleop_key
```



In this terminal, you can control the movement of one of the little turtles by pressing the up, down, left and right keys on the keyboard, and then the other little turtle will follow the movement until they overlap.