LAB4 DELIVERABLES

Jason Zhai

It contains everything for LAB4

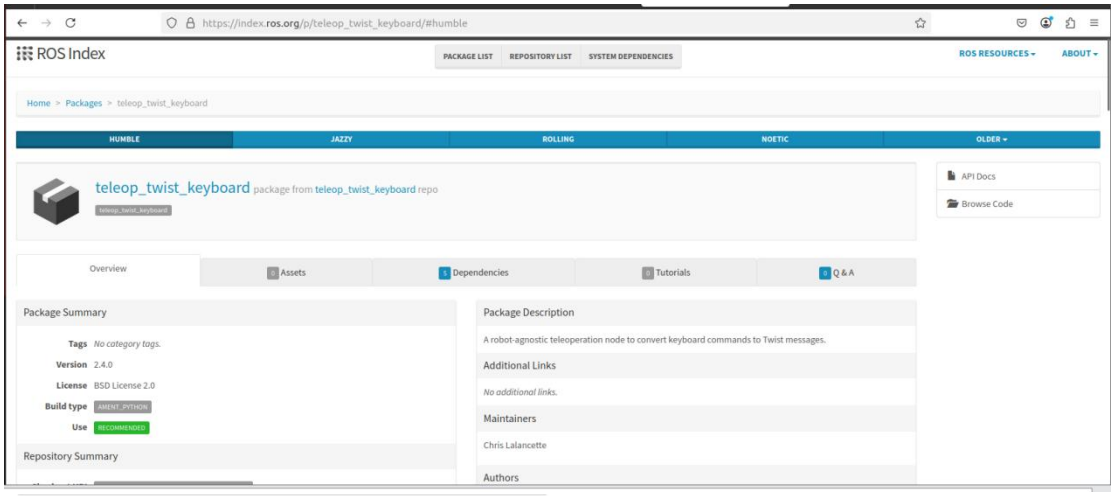
The specified deliverables are highlighted with large red square

Lab # 4

Labs are expected to be completed by each team member. Collaboration is encouraged. The deliverable will be a single report with proof of each member's completion of the tasks in the appendices.

Task 1 - Add an Existing Package

1. **Note:** Let's look for new package to add to pair with the python_turtle package of the last lab. In the current version, to make the turtle move in a different way, we need to modify the client file. It would be useful to add a way to control the motion using the keyboard. This seems like a common desire, so let's see if a package has already been made.
2. **Action:** Go to https://index.ros.org/. In the search bar, type: `name: *keyboard*`.
3. **Note:** The * on either side of key is a wildcard. It tells the search that we are looking for the keyword keyboard with anything on either side of the word which is listed in the name.



Usage

```
This node takes keypresses from the keyboard and publishes them as Twist
messages. It works best with a US keyboard layout.
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit
```

Parameters

- `stamped (bool, default: false)`
  - If false (the default), publish a `geometry_msgs/msg/Twist` message. If true, publish a `geometry_msgs/msg/TwistStamped` message.
- `frame_id (string, default: '')`
  - When `stamped` is true, the frame_id to use when publishing the `geometry_msgs/msg/TwistStamped` message.

7. **Action:** In the Repository Summary in the Overview tab, select the Github link next to Checkout URI.
8. **Action:** Select the green Code button and copy the HTTPS. In the src folder of roscourse_ws, run `git clone` with the copied URL.
9. **Action:** Build this new package.

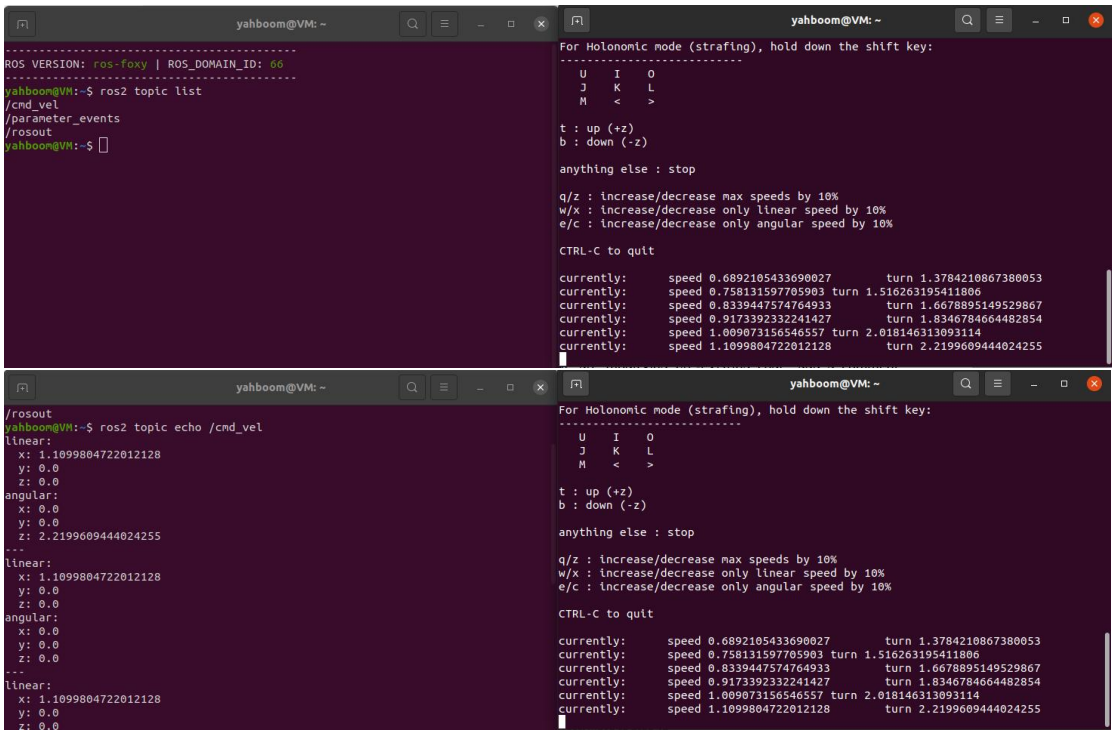
10. **Note:** If we test this new package, we will see that it publishes to the topic 'cmd_vel'. If we look at our python_turtle code, we can see that the topic Twist messages are posted to is different. Let's modify this code to work directly with our python_turtle.

```
yahboom@VM:~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws/src$ git clon
e https://github.com/ros2/teleop_twist_keyboard.git
Cloning into 'teleop_twist_keyboard'...
remote: Enumerating objects: 225, done.
remote: Counting objects: 100% (79/79), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 225 (delta 72), reused 61 (delta 61), pack-reused 146 (from 2)
Receiving objects: 100% (225/225), 43.03 KiB | 2.87 MiB/s, done.
Resolving deltas: 100% (131/131), done.
yahboom@VM:~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws/src$ ls
python_turtle  teleop_twist_keyboard  turtle_interfaces  webcam
```

```
yahboom@VM:~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws/src$ cd ..
yahboom@VM:~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws$ colcon build
[0.695s] WARNING:colcon.colcon_core.package_selection:Some selected packages are already built in one or more underlay workspaces:
        'teleop_twist_keyboard' is in: /opt/ros/foxy
If a package in a merged underlay workspace is overridden and it installs headers, then all packages in the overlay must sort their include directories by workspace order. Failure t
o do so may result in build failures or undefined behavior at run time.
If the overridden package is used by another package in any underlay, then the overriding package in the overlay must be API and ABI compatible or undefined behavior at run time may
 occur.

If you understand the risks and want to override a package anyways, add the following to the command line:
        --allow-overriding teleop_twist_keyboard

This may be promoted to an error in a future release of colcon-override-check.
Starting >>> turtle_interfaces
Starting >>> teleop_twist_keyboard
Starting >>> webcam
Finished <<< teleop_twist_keyboard [1.30s]
Finished <<< webcam [1.64s]
Finished <<< turtle_interfaces [2.36s]
Starting >>> python_turtle
Finished <<< python_turtle [1.29s]

Summary: 4 packages finished [3.96s]
```

```
yahboom@VM:~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws$ ros2 pkg list | grep teleop_twist_keyboard
teleop_twist_keyboard
yahboom@VM:~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws$ ros2 run teleop_twist_keyboard teleop_twist_keyboard

This node takes keypresses from the keyboard and publishes them
as Twist/TwistStamped messages. It works best with a US keyboard layout.
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
currently:      speed 0.55      turn 1.1
currently:      speed 0.6050000000000001        turn 1.2100000000000002
currently:      speed 0.6655000000000002        turn 1.3310000000000004
currently:      speed 0.7320500000000003        turn 1.4641000000000006
```

11. Action: Verify the previous statement using ros2 commands.



12. **Action:** Verify the name of the topic Twists are published to in the python_turtle package. Within `teleop_twist_keyboard.py`, change the topic name on line 151.

13. **Action:** Since we are modifying pre-existing code, add a comment indicated who changed it, when, why, and what the original was.

```
def vels(speed, turn):
    return 'currently:\tspeed %s\tturn %s ' % (speed, turn)


def main():
    settings = saveTerminalSettings()

    rclpy.init()

    node = rclpy.create_node('teleop_twist_keyboard')

    # parameters
    stamped = node.declare_parameter('stamped', False).value
    frame_id = node.declare_parameter('frame_id', '').value
    if not stamped and frame_id:
        raise Exception("'frame_id' can only be set when 'stamped' is True")

    if stamped:
        TwistMsg = geometry_msgs.msg.TwistStamped
    else:
        TwistMsg = geometry_msgs.msg.Twist
   #i changed the topic  it was cmd_vel
    pub = node.create_publisher(TwistMsg, 'turtleDrive', 10)

    spinner = threading.Thread(target=rclpy.spin, args=(node,))
    spinner.start()

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell   ^_ Go To Li
```

14. **Action:** In the python_turtle package, within turtlebot_client.py, comment out the line where we publish the command velocity in the main function.

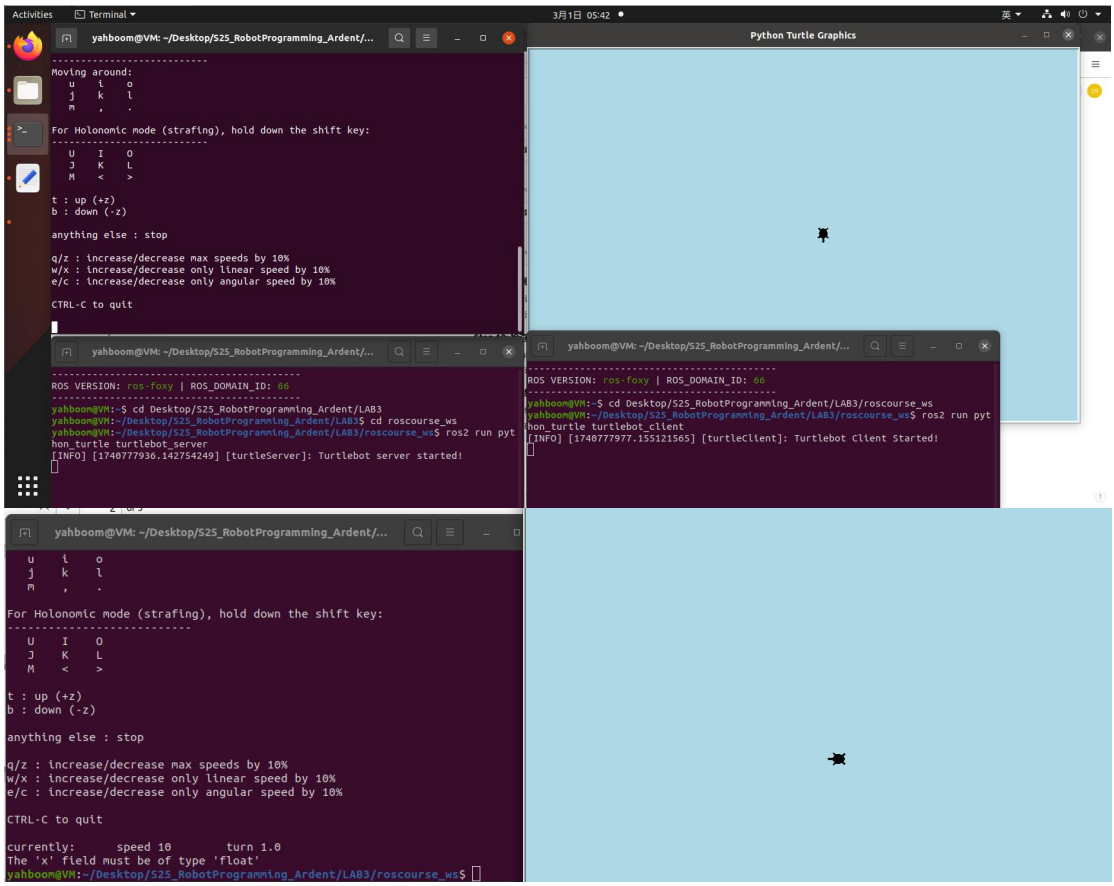
```
    while rclpy.ok():

        cli_obj.update()
        rclpy.spin_once(cli_obj)

        unit_x = 1 #<put a reasonable ratio, 1 is a good number, around 1 is good enough>
        unit_z = 1 #<put a reasonable ratio, 1 is a good number, around 1 is good enough>

        #### publish twist ####
        cmd_msg = Twist()
        cmd_msg.linear.x = float(50 * unit_x)
        cmd_msg.angular.z = float(1 * unit_z)
#       cli_obj.twist_pub.publish(cmd_msg)

    # Destory the node explicitly
    cli_obj.destroy_node()
    rclpy.shutdown()

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

15. **Note:** We will be further modifying this file. You could create a copy without the edits called turtlebot_client_old.py and add an entry_point in the setup.py if you want to keep the original as a reference.

16. **Action:** Rebuild the teleop_twist_keyboard and python_turtle packages.

17. **Action:** In three separate terminals run the turtlebot server, the turtlebot client, and the teleop node. You should now be able to drive your turtlebot using the keyboard.

    - **Question:** What do you observe when trying to move the turtle?
    - **Action:** Take a look at line 157 in teleop_twist_keyboard.py. Modify this to address the issue observed.
    - **Question:** What happens if you set the value of speed to 10 instead of 10.0? Why?



Task 2 - Adding Parameters

1. **Note:** Let's add a parameter to our turtlebot_client which will allow us to initialize the color of the turtle when we start up the node.
2. **Action:** Add a dependency to rcl_interfaces in the package.xml file.
3. **Action:** Within `turtlebot_client.py` add the following imports

- `from rclpy.parameter import Parameter`
- `from rcl_interfaces.msg import ParameterDescriptor`

Open — **package.xml** — ~/Desktop/S25_RobotProgramming_Ardent/LAB3/roscourse_ws/src/python_turtle

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>python_turtle</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="jzhai87@gmail.com">yahboom</maintainer>
  <license>TODO: License declaration</license>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
  <depend>rclpy</depend>
  <depend>turtle_interfaces</depend>
  <depend>rcl_interfaces</depend>

</package>
```

Open — **turtlebot_client.py** — ~/Desktop/S25_RobotProgramming_Ardent/LA...ourse_ws/src/python_turtle/python_turtle

```
import rclpy
from rclpy.node import Node
import math
import random

import turtle

from geometry_msgs.msg import Twist, Pose

from turtle_interfaces.srv import SetColor
from turtle_interfaces.msg import TurtleMsg

from rclpy.parameter import Parameter
from rcl_interfaces.msg import ParameterDescriptor

class TurtleClient(Node):
    def __init__(self):
        super().__init__('turtleClient')

        #### Display/Turtle Setup ####
        self.screen = turtle.Screen()
        self.screen.bgcolor('lightblue')
        self.turtle_display = turtle.Turtle()
        self.turtle_display.shape("turtle")
        self.turtle = TurtleMsg()
```

4. **Action:** Add the parameter in the init method of the TurtleClient class.
    - `self.declare_parameter('turtleColor', '<default_color>', ParameterDescriptor(description= '<description>'))`
    - **Note:** You should fill in the default color and description.
5. **Note:** The parameter needs to set the turtle color on startup. Let's set that up next.
6. **Action:** Read the parameter value and change the turtle display color in the init method.
    - `turtleColor = self.get_parameter('turtleColor').get_parameter_value().string_value`
    - `self.turtle_display.color(turtleColor)`
7. **Question:** What other value types are available when we get the paramter value?

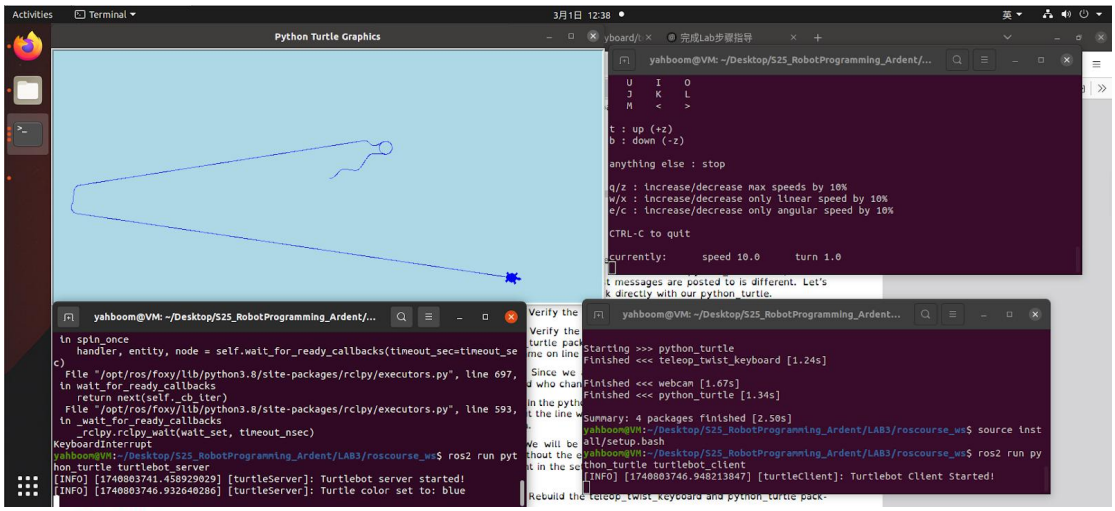
8. **Note:** If you build and run the three nodes: turtlebot_server, turtlebot_client, teleop_twist_keyboard, the turtle will appear as the default color. However, when you drive around the turtle, there is no line. Why?

9. **Note:** Take a look at turtlebot_server.py. Notice that it generates a blank TurtleMsg in the init method that it publishes to the topic 'turtleState'. In our client, we can see that it is subscribed to this topic. Therefore, the self.turtle which is a TurtleMsg gets its field values from the server. Since the server is publishing a blank field for the color, the client's update function sees 'None' for self.turtle.color and runs the penup method for the turtle display.

10. **Note:** So this tells us that we need to make sure the server has the desired field for the color. We can achieve this by calling the setColor service in our client initialization.

```
26
27        #### publisher define ####
28        self.twist_pub = self.create_publisher(Twist, 'turtleDrive', 1)
29        ############################
30
31        #### subscribing turtlebot state ####
32        self.turtle_sub = self.create_subscription(TurtleMsg, 'turtleState', self.turtle_callback, 1)
33
34        #### setting the turtle color to blue as default ####
35        self.declare_parameter('turtleColor', 'blue', ParameterDescriptor(description="Color of the turtle"))
36        turtleColor = self.get_parameter('turtleColor').get_parameter_value().string_value
37        self.turtle_display.color(turtleColor)
38
39        self.color_cli = self.create_client(SetColor, 'set_color')
40        while not self.color_cli.wait_for_service(timeout_sec=1.0):
41            self.get_logger().info('Color service not available, waiting...')
42        self.color_req = SetColor.Request()
43        self.color_req.color = self.get_parameter('turtleColor').get_parameter_value().string_value
44        self.server_call = True
45
46
```

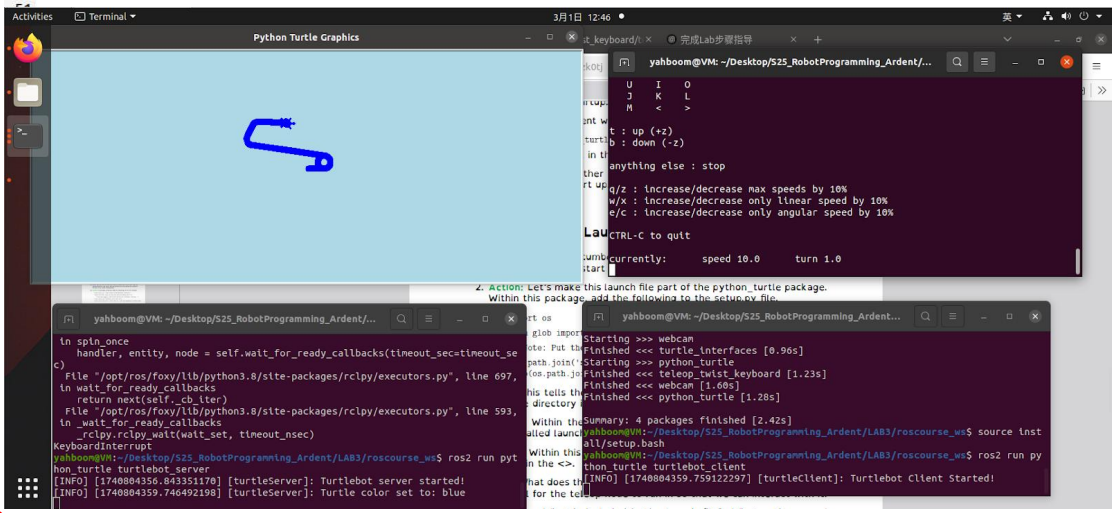
12. **Action:** Build and source your terminals. You should now be able to run the server, client, and teleop without the server overriding your chosen color on startup.



13. **Action:** Run the client with a non-default parameter color.
    - `ros2 run python_turtle turtlebot_client --ros-args --param <name>:=<value>`
    - You need to fill in the name and value.

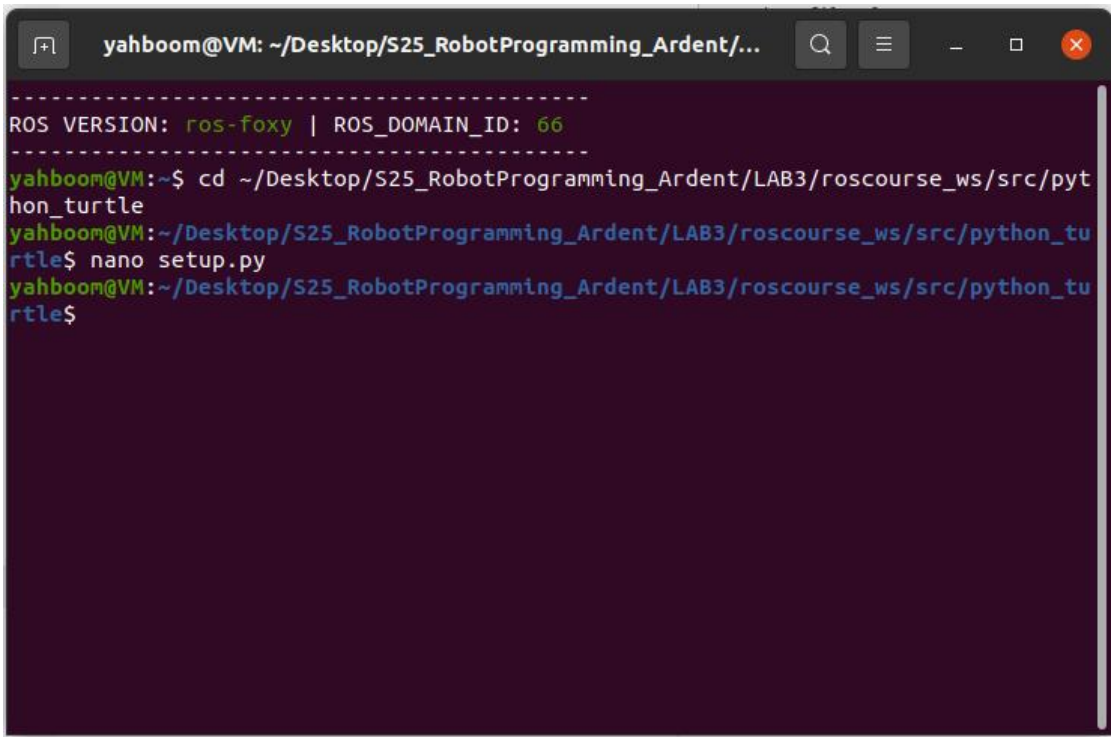
14. **Action:** Create another parameter in the turtlebot client which sets the pen size on start up.

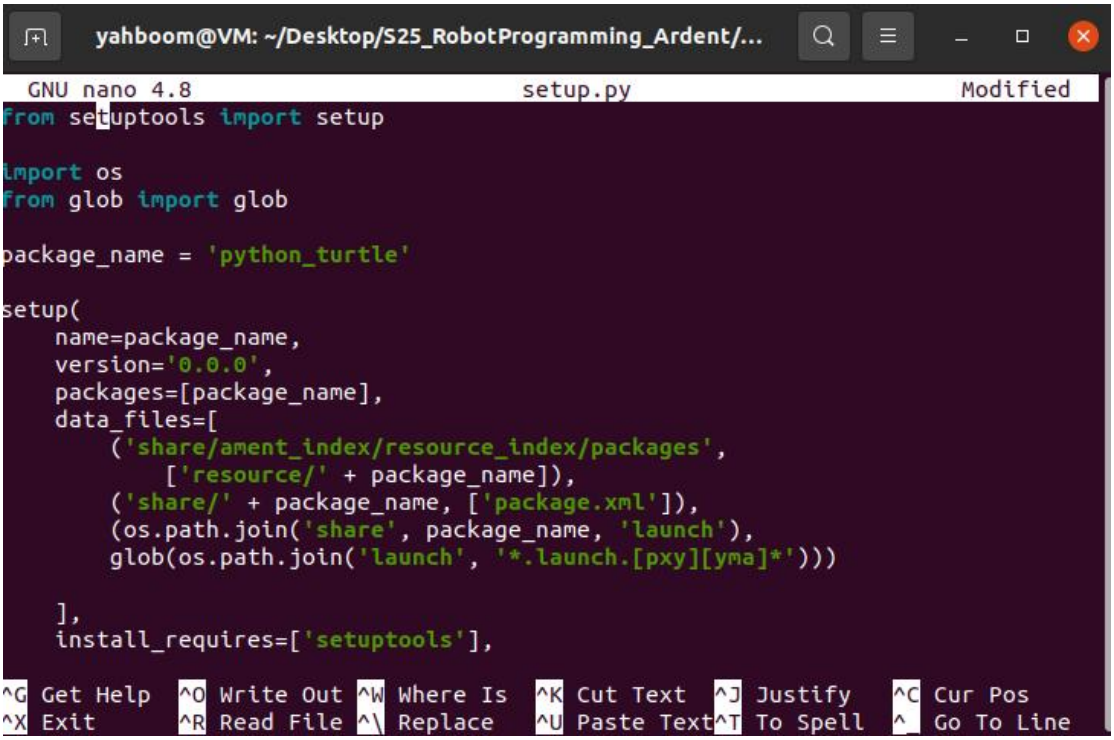
```
33
34      #### setting the turtle color to blue as default ####
35      self.declare_parameter('turtleColor', 'blue', ParameterDescriptor(description="Color of the turtle"))
36      turtleColor = self.get_parameter('turtleColor').get_parameter_value().string_value
37      self.turtle_display.color(turtleColor)
38
39      self.declare_parameter('penSize', 10, ParameterDescriptor(description="Size of the Pen"))
40      penSize = self.get_parameter('penSize').get_parameter_value().integer_value
41      self.turtle_display.pensize(penSize)
42
43      self.color_cli = self.create_client(SetColor, 'set_color')
44      while not self.color_cli.wait_for_service(timeout_sec=1.0):
45          self.get_logger().info('Color service not available, waiting...')
46      self.color_req = SetColor.Request()
47      self.color_req.color = self.get_parameter('turtleColor').get_parameter_value().string_value
48      self.server_call = True
49      self.service_future = self.color_cli.call_async(self.color_req)
50
```



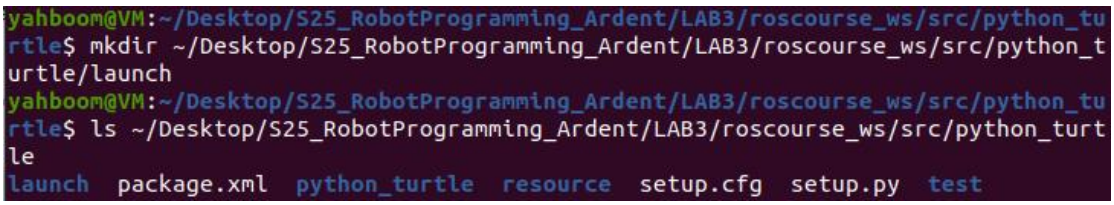
Task 3 - Create Launch Files

1. **Note:** It gets a bit cumbersome having so many terminals. Let's create a launch file to start our server, client, and teleop nodes together.
2. **Action:** Let's make this launch file part of the python_turtle package. Within this package, add the following to the setup.py file.
   - `import os`
   - `from glob import glob`
   - `** Note: Put the following within the data_files square brackets **`
   - `(os.path.join('share',package_name,'launch'), glob(os.path.join('launch','*launch.[pxy][yma]*')))`
3. **Note:** This tells the build process to look for launch files within this package directory inside a folder called launch.





4. **Action:** Within the python_turtle package main folder, add a new folder called launch.



5. **Action:** Within this folder copy the file called `turtle_teleop_launch.py` and fill in the <>.

yahboom@VM: ~/Desktop/S2S_RobotProgramming_Ardent/LAB3/roscourse_ws/src/python_turtle/launch

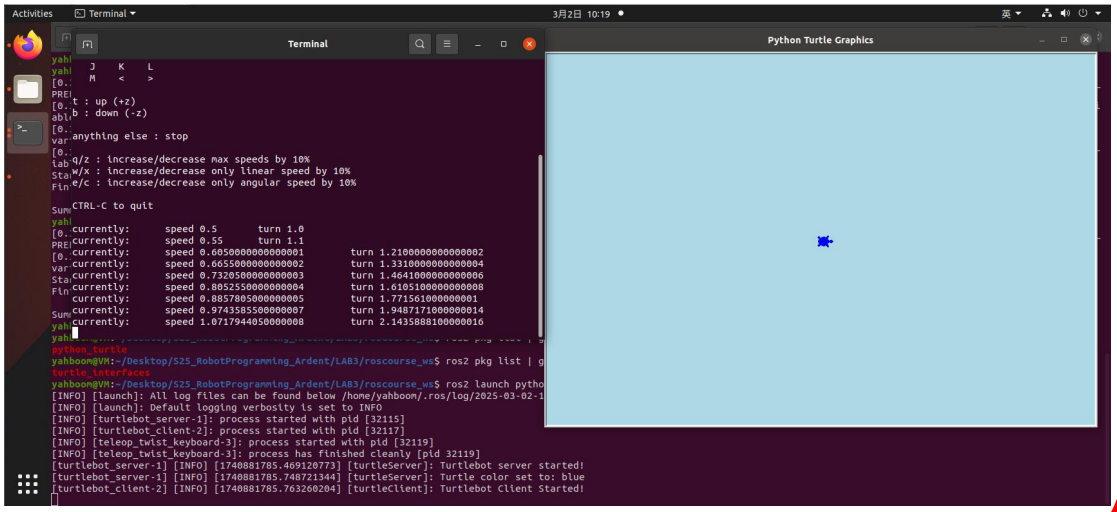
GNU nano 4.8 turtle_teleop_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node  # <-- FILL MISSING PART

def generate_launch_description():
    server = Node(
        package='python_turtle',  # <-- FILL MISSING PART
        executable='turtlebot_server'
    )

    client = Node(
        package='python_turtle',
        executable='turtlebot_client',
        parameters=[{'turtleColor': 'blue'}]  # <-- FILL MISSING PART
    )

    teleop = Node(
        package='teleop_twist_keyboard',
        executable='teleop_twist_keyboard',
        output='screen',
        prefix='gnome-terminal --'
    )

    return LaunchDescription([
        server,
        client,  # <-- FILL MISSING PART
        teleop
    ])
```

6. **Note:** What does the prefix in the teleop node do? It opens a separate terminal for the teleop node to run in so that we can interact with it.

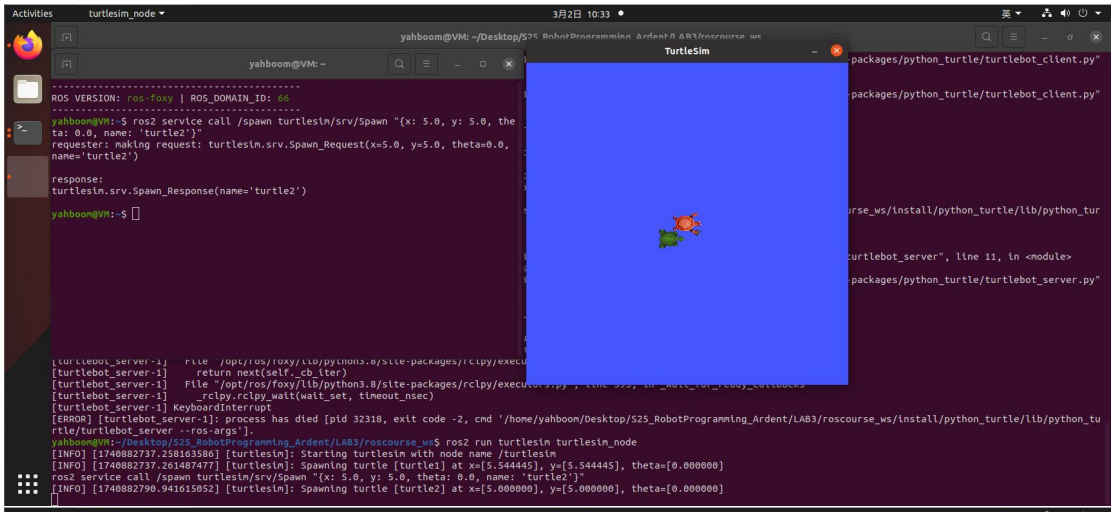
7. **Question:** What is included in the launch file? What options can I define for a node via a launch file?

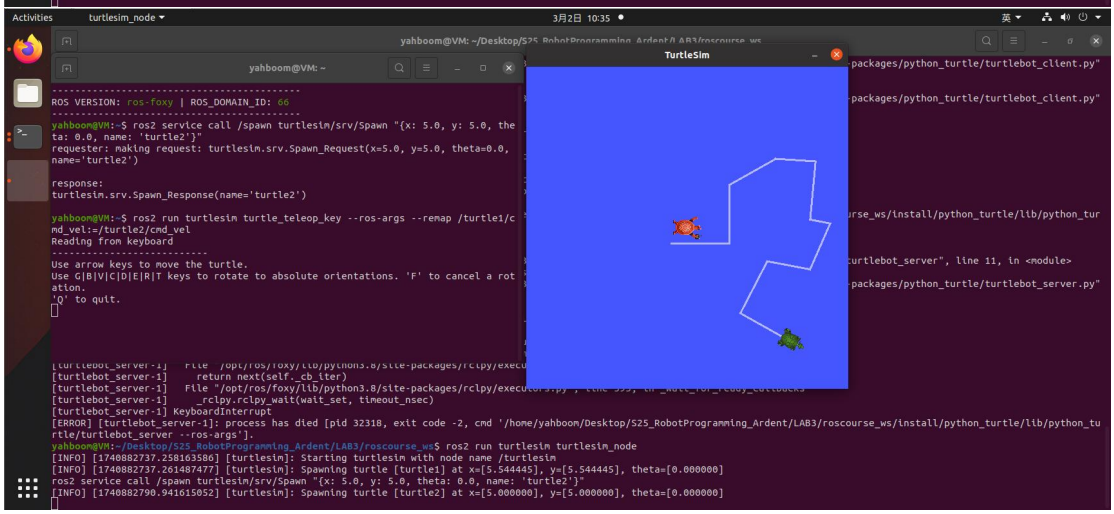
8. **Action:** Build and source. Now run everything with the command `ros2 launch python_turtle turtle_teleop_launch.py`

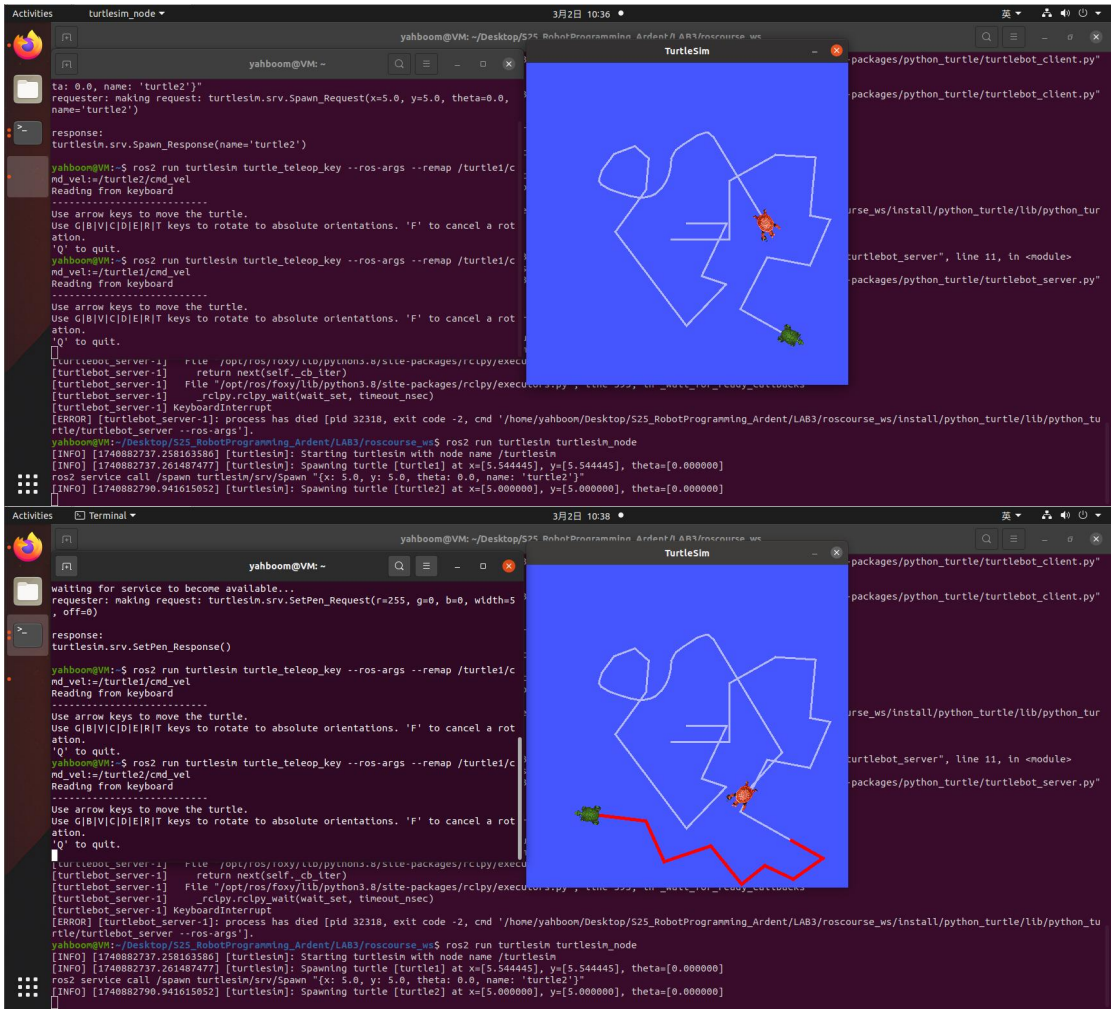


Task 4 - Remapping

1. **Action:** Execute the turtlesim node from the turtlesim package (the one used in lab 2).
2. **Action:** Use the spawn command to generate a new turtle.
   - `ros2 service call /spawn turtlesim/srv/Spawn "{name: '<name>'}"`
3. **Action:** To control this turtle, run the teleop command with remapping.
   - `ros2 run turtlesim turtle_teleop_key --ros-args --remap /turtle1/cmd_vel:=/<name>/cmd_vel`
4. **Note:** To make it look interesting, change the pen color of each turtle by calling the appropriate service.
5. **Question:** Why is the ability to do this remapping so useful?







Remapping allows reconfiguring ROS nodes without modifying code. It enables dynamic control of multiple robots, prevents topic conflicts, simplifies debugging, and enhances modularity by making nodes reusable in different environments.