Robot information release

Note: The ROS_DOMAIN_ID of the Raspberry Pi and the microROS control board need to be consistent. You can check [MicroROS Control Board Parameter Configuration] to set the microROS control board ROS_DOMAIN_ID. Check the tutorial [Connect MicroROS Agent] to determine whether the IDs are consistent.

1. Program function description

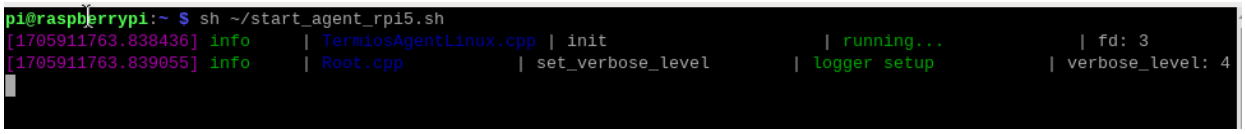
After the car is connected to the agent, it will publish sensor data such as radar and imu. You can run commands in the supporting virtual machine/Raspberry Pi 5 to query this information. You can also publish control data of sensors such as speed and buzzer.

2. Query car information

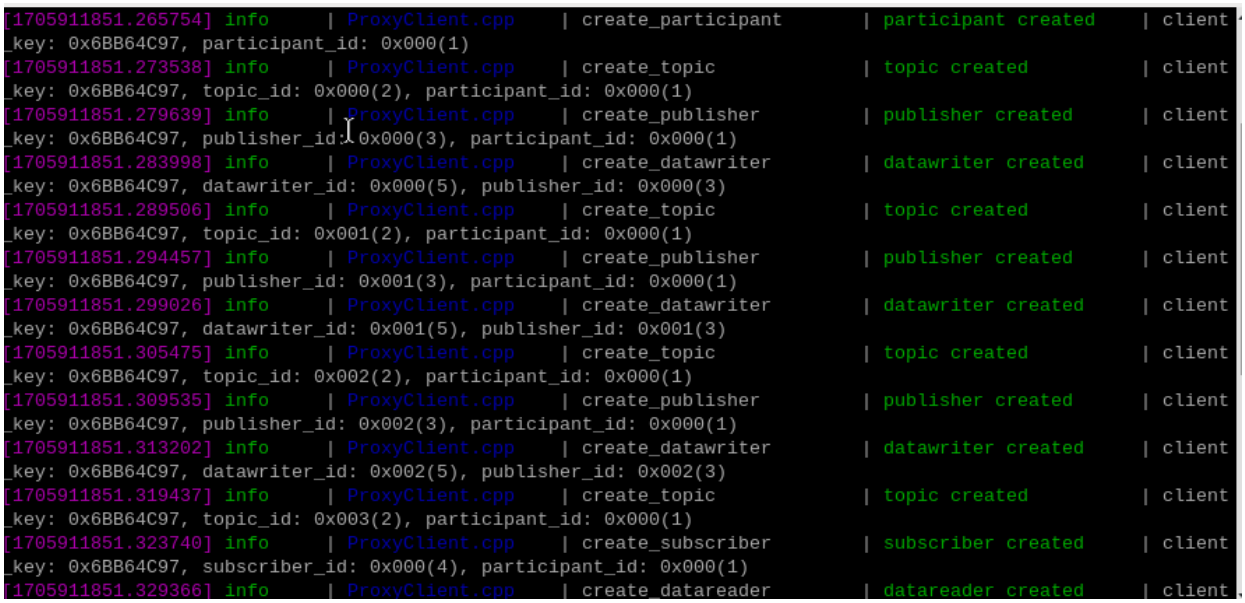
2.1. Start and connect to the agent

After successfully starting the Raspberry Pi, enter the following command to start the agent.

```
sh ~/start_agent_rpi5.sh
```



Then, turn on the car switch and wait for the car to connect to the agent. The connection is successful, as shown in the figure below.

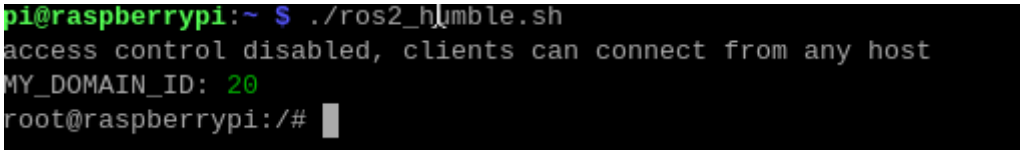


2.2. Enter the car docker

Open another terminal and enter the following command to enter docker.

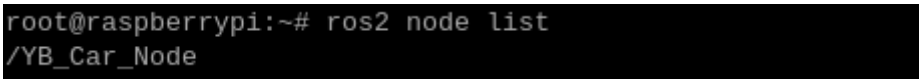
```
sh ros2_humble.sh
```

When the following interface appears, you have successfully entered docker. Now you can control the car through commands.



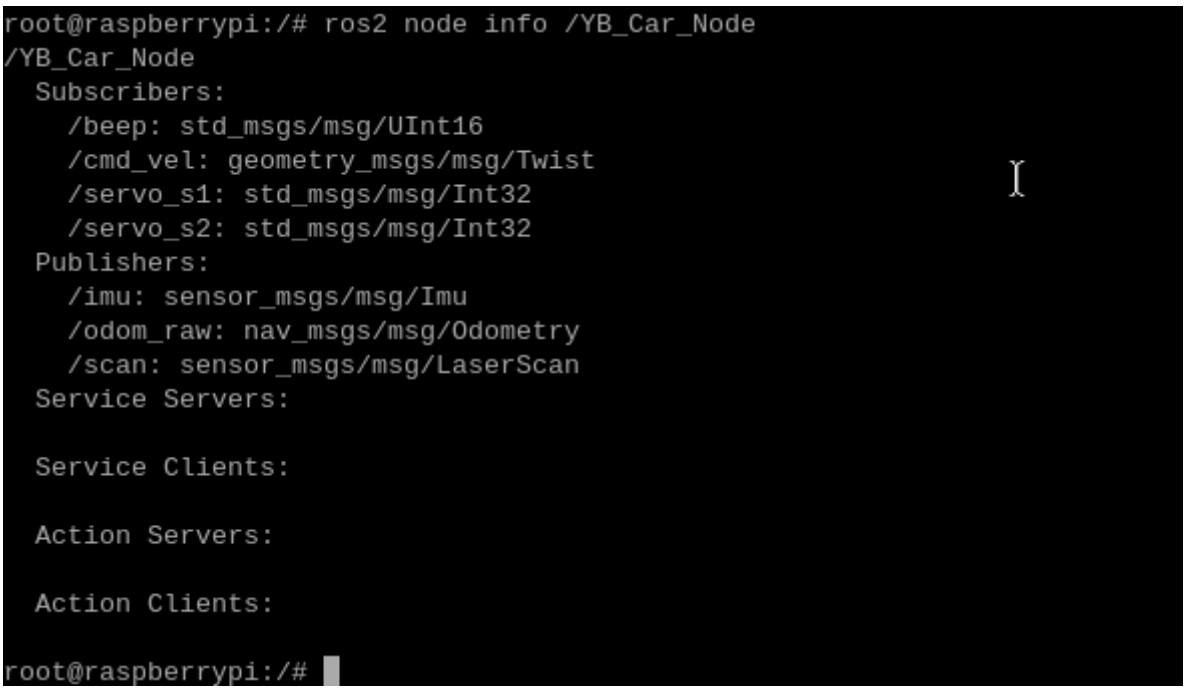
Enter the following command in the terminal to query the agent node.

```
ros2 node list
```



Then enter the following command to query which topics the node has published/subscribed to,

```
ros2 node info /YB_Car_Node
```



It can be seen that the topics subscribed are as follows

/beep：Buzzer control

/cmd_vel：Car speed control

/servo_s1：s1 servo gimbal control

/servo_s2：s2 servo gimbal control

Posted topics include:

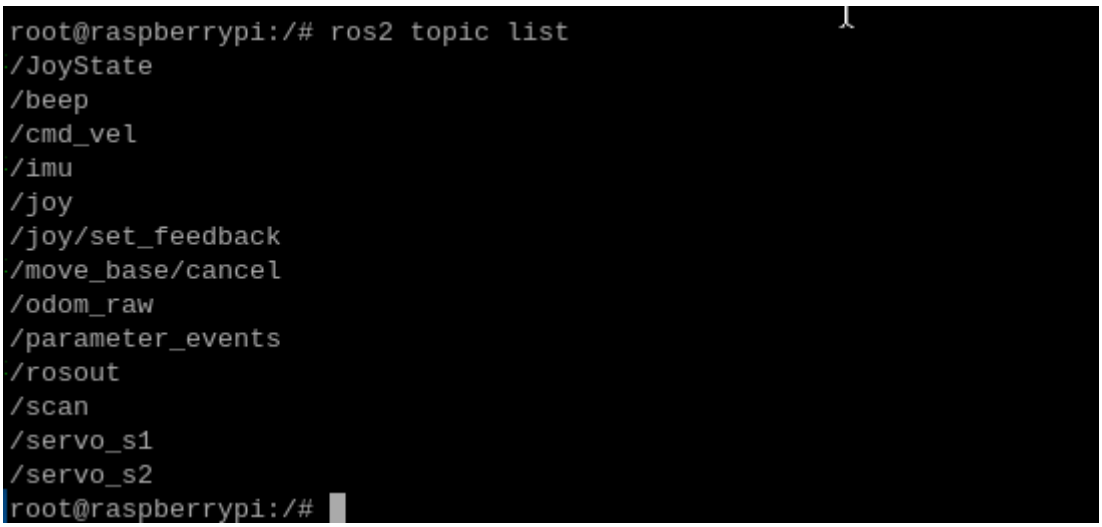
/imu：imu module data

/odom：Odometer module data

/scan：Radar module data

We can also query the topic command and enter it in the terminal.

```
ros2 topic list
```



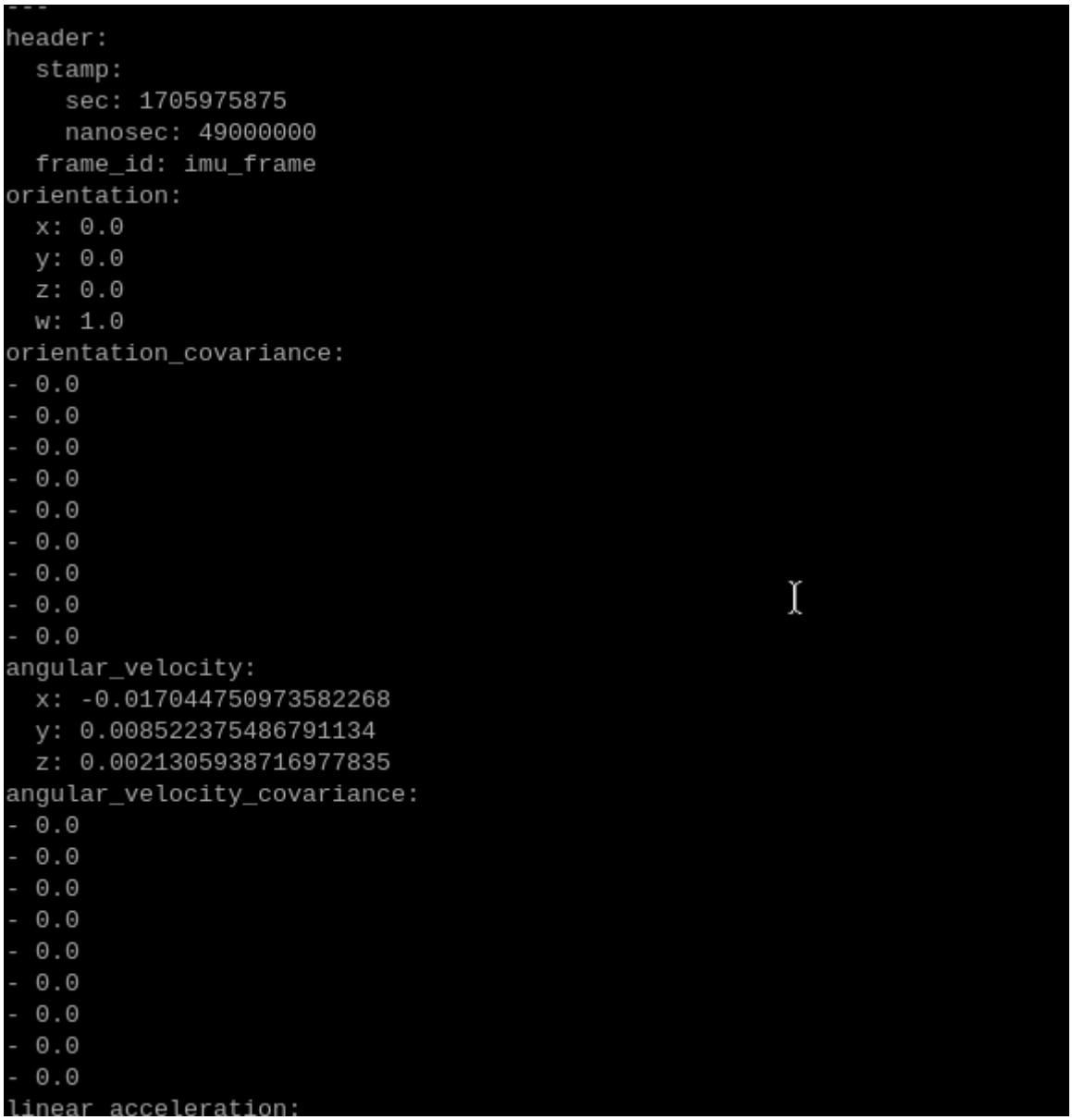
2.3. Query topic data

Query radar data.

```
ros2 topic echo /scan
```

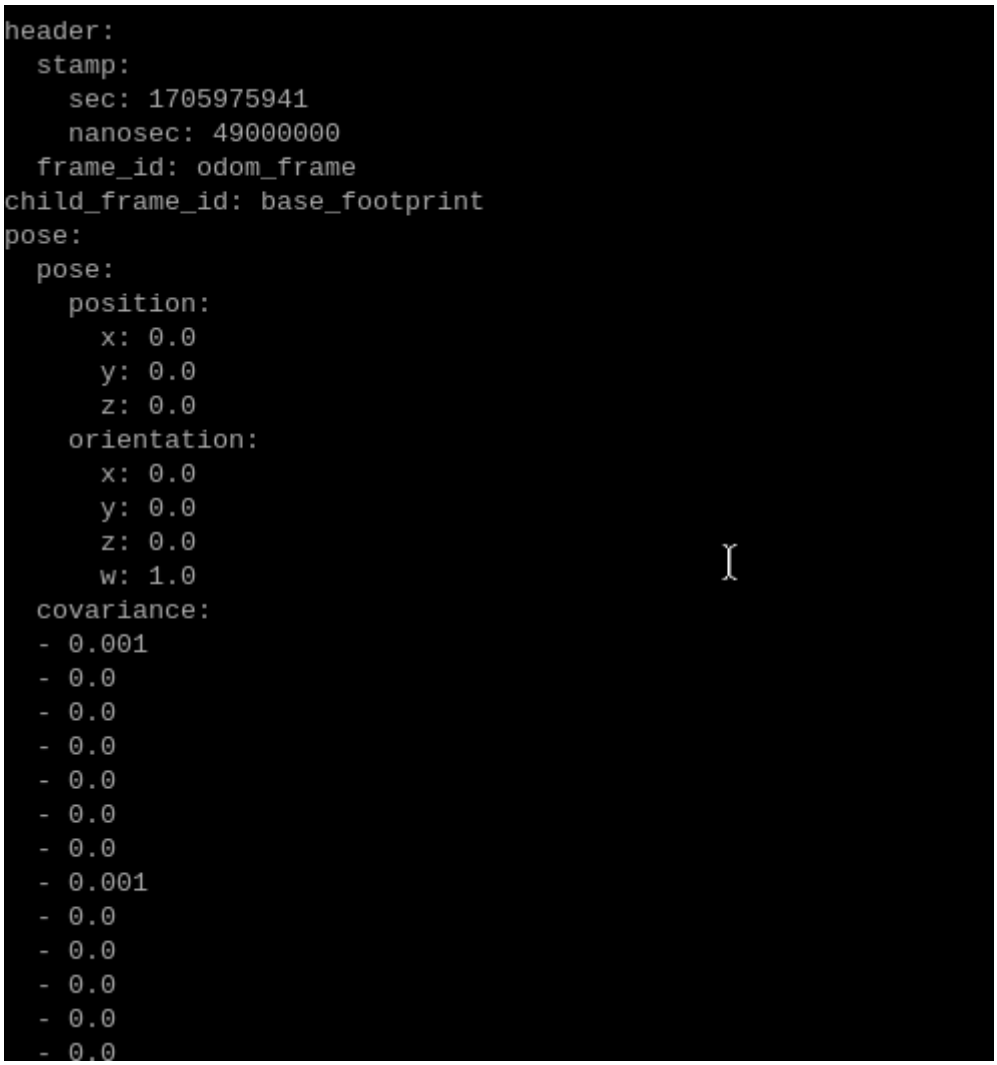
Query imu data.

```
ros2 topic echo /imu
```



Query odom data.

```
ros2 topic echo /odom_raw
```

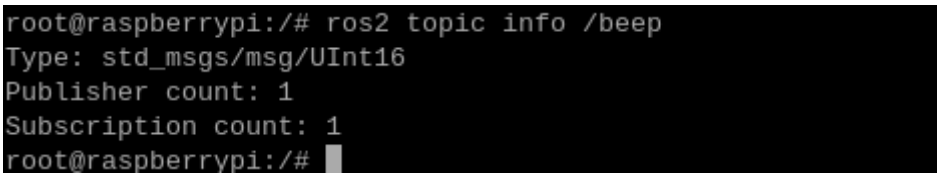


3. Publish car control information

3.1. Control the buzzer

First, query the relevant information about the following buzzer topics and enter it in the terminal.

```
ros2 topic info /beep
```



Learn that the data type is std_msgs/msg/UInt16. Then enter the following command to turn on the buzzer and enter in the terminal.

```
ros2 topic pub /beep std_msgs/msg/UInt16 "data: 1"
```

```
root@raspberrypi:/# ros2 topic pub /beep std_msgs/msg/UInt16 "data: 1"
publisher: beginning loop
publishing #1: std_msgs.msg.UInt16(data=1)

publishing #2: std_msgs.msg.UInt16(data=1)

publishing #3: std_msgs.msg.UInt16(data=1)

publishing #4: std_msgs.msg.UInt16(data=1)

publishing #5: std_msgs.msg.UInt16(data=1)
```

Enter the following command to turn off the buzzer, terminal input.

```
ros2 topic pub /beep std_msgs/msg/UInt16 "data: 0"
```

```
root@raspberrypi:/# ros2 topic pub /beep std_msgs/msg/UInt16 "data: 0"0"
publisher: beginning loop
publishing #1: std_msgs.msg.UInt16(data=0)

publishing #2: std_msgs.msg.UInt16(data=0)

publishing #3: std_msgs.msg.UInt16(data=0)

publishing #4: std_msgs.msg.UInt16(data=0)

publishing #5: std_msgs.msg.UInt16(data=0)
```

3.2. Release speed control information

We assume that the released car moves at a linear speed of 0.5 and an angular speed of 0.2, and the terminal inputs instructions.

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.2}}"
```

```
root@raspberrypi:/# ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{lin
ear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.2}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.5, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
2))

publishing #2: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.5, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
2))

publishing #3: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.5, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
2))

publishing #4: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.5, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
2))
```

Set the car speed to zero

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0},
angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

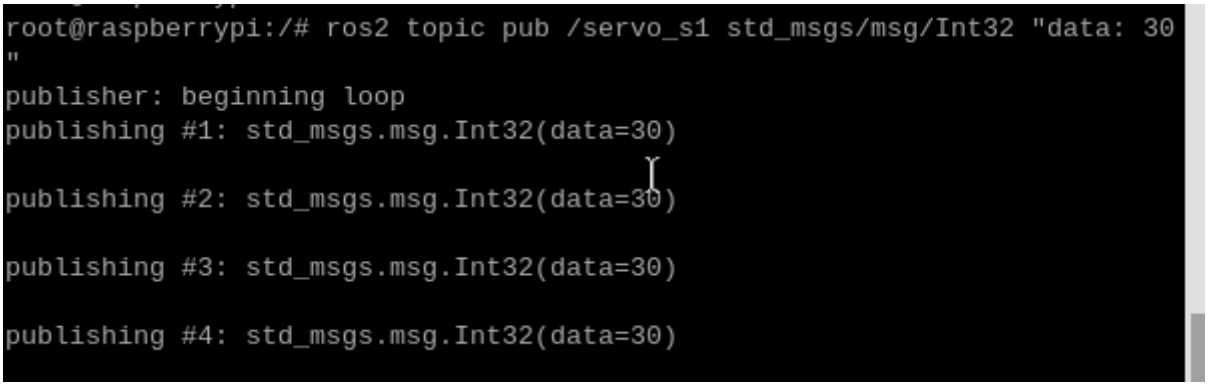
```
root@raspberrypi:/# ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{lin
ear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
0))

publishing #2: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
0))

publishing #3: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
0))

publishing #4: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
0))

publishing #5: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x
=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.
```

3.3. Control the gimbal servo

What needs to be noted here is that the range of s1 servo is [-90,90], and the range of s2 servo is [-90,20]. If the value exceeds the range, the servo will not rotate.

We assume that the s1 servo is controlled to rotate 30 degrees, and the terminal input is,

```
ros2 topic pub /servo_s1 std_msgs/msg/Int32 "data: 30"
```



In the same way, to control the s2 servo to rotate -30 degrees, enter the following command on the terminal:

```
ros2 topic pub /servo_s2 std_msgs/msg/Int32 "data: -30"
```