ROS Control

1. Course Content

1. Learn the basics of robot control using ROS.

This function enables control of the robot's speed, buzzer, and robotic arm using ROS2 topic tools. It also enables reading low-level data, such as radar data, IMU data, and odometry data.

2. Preparation

2.1 Content Description

This course uses the Jetson Orin NX as an example. For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this course in the terminal. For instructions on entering the Docker container from the host computer, refer to the **[Configuration and Operation Guide] -- [Entering the Docker (Jetson Nano and Raspberry Pi 5 users see here)]** section of this product tutorial. For Orin and NX boards, simply open a terminal and enter the commands mentioned in this course.

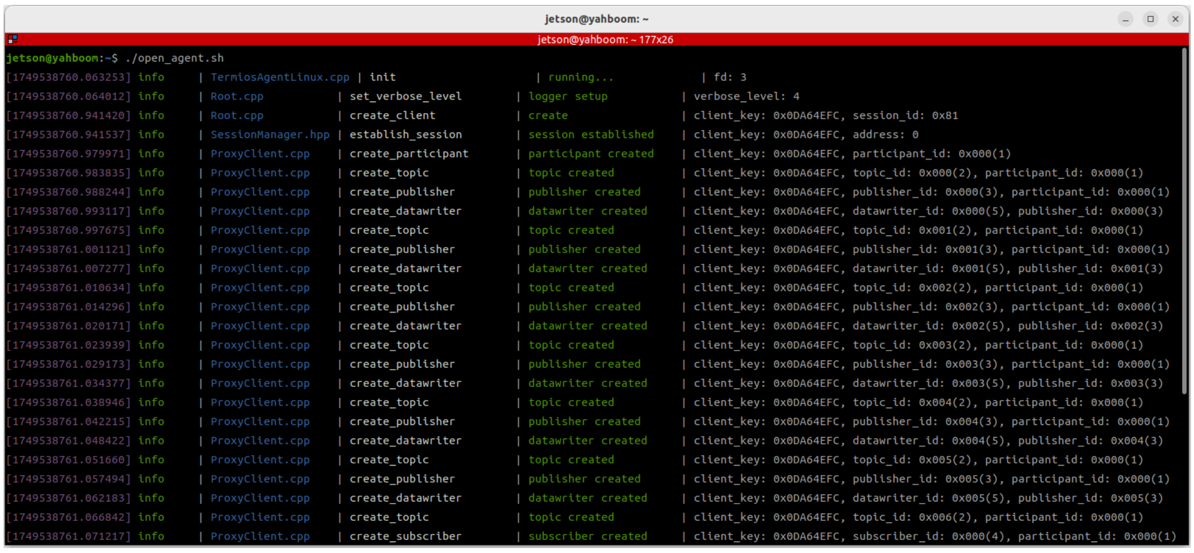
2.2 Starting the Agent

Note: The Docker agent must be started before testing all cases. If it's already started, you don't need to restart it.

Enter the following command in the vehicle terminal:

```
sh start_agent.sh
```

The terminal will print the following message, indicating a successful connection.



```
jetson@yahboom:~$ ./open_agent.sh
[1749538760.063253] info     | TermiosAgentLinux.cpp | init                     | running...             | fd: 3
[1749538760.064012] info     | Root.cpp            | set_verbose_level        | logger setup           | verbose_level: 4
[1749538760.941420] info     | Root.cpp            | create_client            | create                 | client_key: 0x0DA64EFC, session_id: 0x81
[1749538760.941537] info     | SessionManager.hpp | establish_session        | session established    | client_key: 0x0DA64EFC, address: 0
[1749538760.979971] info     | ProxyClient.cpp     | create_participant       | participant created    | client_key: 0x0DA64EFC, participant_id: 0x000(1)
[1749538760.983835] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x000(2), participant_id: 0x000(1)
[1749538760.988244] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x000(3), participant_id: 0x000(1)
[1749538760.993117] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x000(5), publisher_id: 0x000(3)
[1749538760.997675] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x001(2), participant_id: 0x000(1)
[1749538761.001121] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x001(3), participant_id: 0x000(1)
[1749538761.007277] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x001(5), publisher_id: 0x001(3)
[1749538761.010634] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x002(2), participant_id: 0x000(1)
[1749538761.014296] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x002(3), participant_id: 0x000(1)
[1749538761.020171] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x002(5), publisher_id: 0x002(3)
[1749538761.023939] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x003(2), participant_id: 0x000(1)
[1749538761.029173] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x003(3), participant_id: 0x000(1)
[1749538761.034377] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x003(5), publisher_id: 0x003(3)
[1749538761.038946] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x004(2), participant_id: 0x000(1)
[1749538761.042215] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x004(3), participant_id: 0x000(1)
[1749538761.048422] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x004(5), publisher_id: 0x004(3)
[1749538761.051660] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x005(2), participant_id: 0x000(1)
[1749538761.057494] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x005(3), participant_id: 0x000(1)
[1749538761.062183] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x005(5), publisher_id: 0x005(3)
[1749538761.066842] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0DA64EFC, topic_id: 0x006(2), participant_id: 0x000(1)
[1749538761.071217] info     | ProxyClient.cpp     | create_subscriber        | subscriber created     | client_key: 0x0DA64EFC, subscriber_id: 0x000(4), participant_id: 0x000(1)
```

3. Startup Commands

3.1 Functional Description

This function enables control of the vehicle's speed, buzzer, and robotic arm through ROS2 topic tools. It also enables reading low-level data, such as radar data, IMU data, and odometer data.

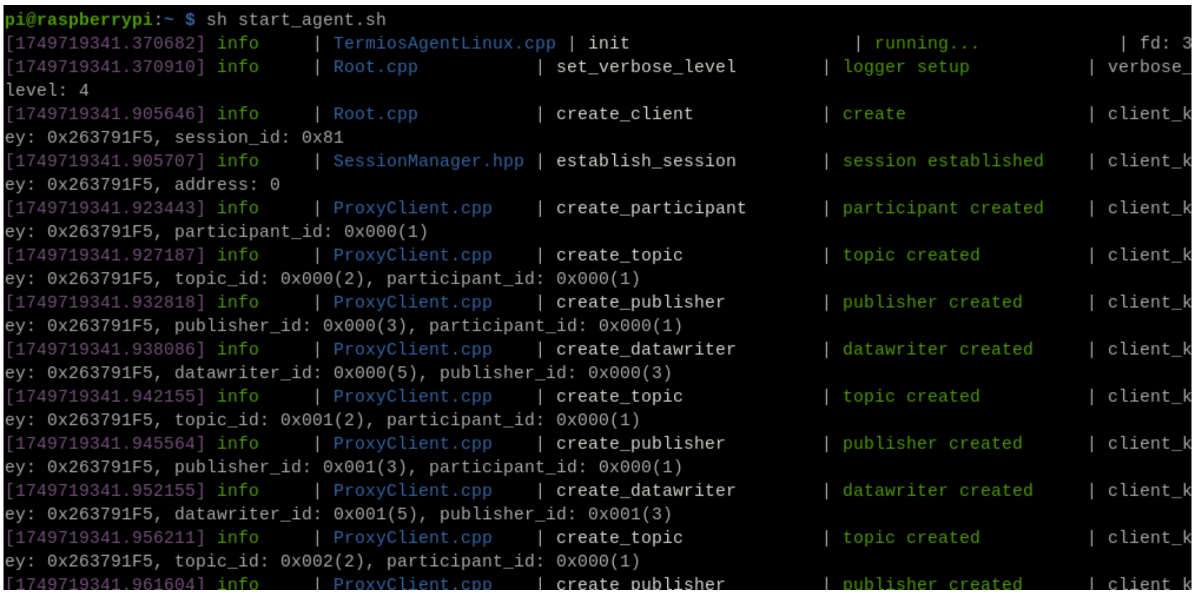
3.2 Program Startup

3.2.1 Connecting to the Agent

After booting up, open a terminal and enter the following command to connect to the agent:

```
sh start_agent.sh
```

As shown below, after successfully starting the agent,



```
pi@raspberrypi:~ $ sh start_agent.sh
[1749719341.370682] info     | TermiosAgentLinux.cpp | init                     | running...             | fd: 3
[1749719341.370910] info     | Root.cpp            | set_verbose_level        | logger setup           | verbose_
level: 4
[1749719341.905646] info     | Root.cpp            | create_client            | create                 | client_k
ey: 0x263791F5, session_id: 0x81
[1749719341.905707] info     | SessionManager.hpp | establish_session        | session established    | client_k
ey: 0x263791F5, address: 0
[1749719341.923443] info     | ProxyClient.cpp     | create_participant       | participant created    | client_k
ey: 0x263791F5, participant_id: 0x000(1)
[1749719341.927187] info     | ProxyClient.cpp     | create_topic             | topic created          | client_k
ey: 0x263791F5, topic_id: 0x000(2), participant_id: 0x000(1)
[1749719341.932818] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_k
ey: 0x263791F5, publisher_id: 0x000(3), participant_id: 0x000(1)
[1749719341.938086] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_k
ey: 0x263791F5, datawriter_id: 0x000(5), publisher_id: 0x000(3)
[1749719341.942155] info     | ProxyClient.cpp     | create_topic             | topic created          | client_k
ey: 0x263791F5, topic_id: 0x001(2), participant_id: 0x000(1)
[1749719341.945564] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_k
ey: 0x263791F5, publisher_id: 0x001(3), participant_id: 0x000(1)
[1749719341.952155] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_k
ey: 0x263791F5, datawriter_id: 0x001(5), publisher_id: 0x001(3)
[1749719341.956211] info     | ProxyClient.cpp     | create_topic             | topic created          | client_k
ey: 0x263791F5, topic_id: 0x002(2), participant_id: 0x000(1)
[1749719341.961604] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_k
```

If the startup fails, check for loose connections and verify that the serial port /dev/myserial is recognized by running `ls /dev.myserial`.

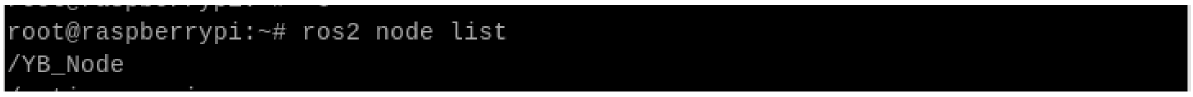
3.2.2 Viewing Node Information

After successfully connecting to the agent, you can use the `ros2 node list` command in the terminal to view the nodes.

Here, you need to open the terminal according to the motherboard. Users of Jetson-Nano and Raspberry Pi 5 need to enter Docker and enter the command. For the instructions for starting Docker, please refer to the content of [Entering Docker (Jetson-Nano and Raspberry Pi 5 users see here)] in [0. Instructions and Installation Steps] of this product tutorial; users of Orin motherboards can directly open the terminal and enter the command. Here, we use the Raspberry Pi 5's runtime interface as an example. After entering Docker, enter the command in the Docker terminal:

```
ros2 node list
```

A screenshot is shown below.

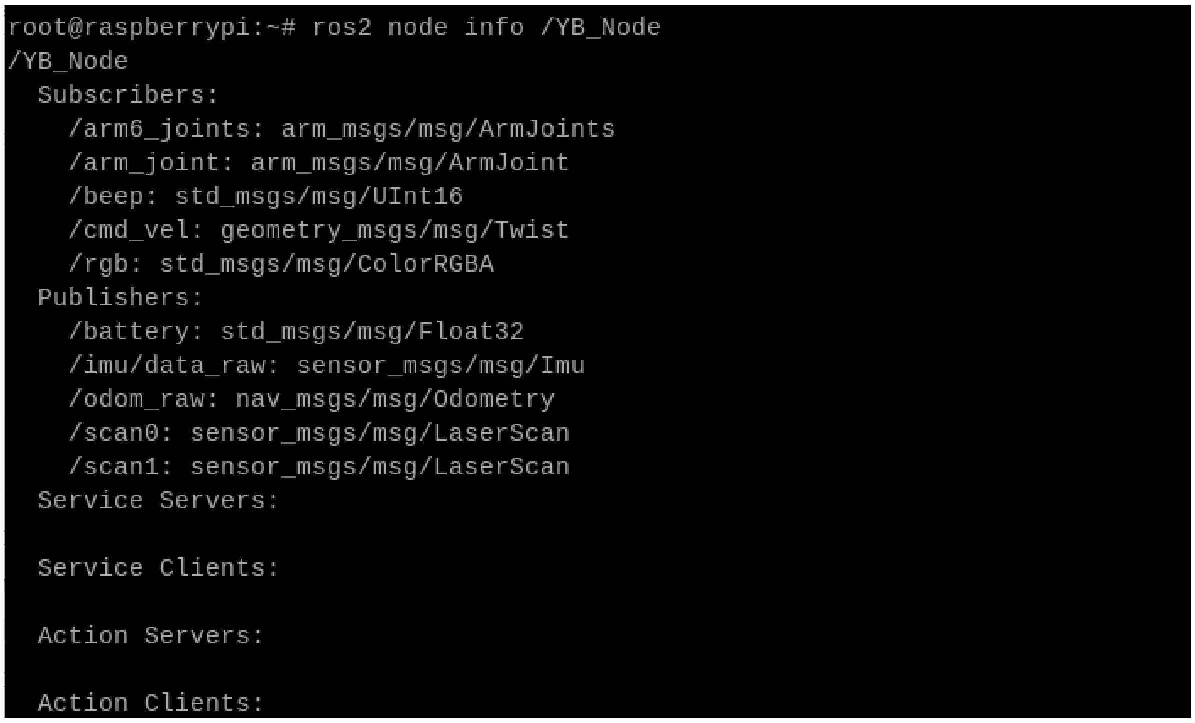


```
root@raspberrypi:~# ros2 node list
/YB_Node
```

Here, there's a /YB_Node, indicating that the underlying control node has started. Enter the following command to query information about this node, also in the terminal:

```
ros2 node info /YB_Node
```

A screenshot is shown below.



```
root@raspberrypi:~# ros2 node info /YB_Node
/YB_Node
  Subscribers:
    /arm6_joints: arm_msgs/msg/ArmJoints
    /arm_joint: arm_msgs/msg/ArmJoint
    /beep: std_msgs/msg/UInt16
    /cmd_vel: geometry_msgs/msg/Twist
    /rgb: std_msgs/msg/ColorRGBA
  Publishers:
    /battery: std_msgs/msg/Float32
    /imu/data_raw: sensor_msgs/msg/Imu
    /odom_raw: nav_msgs/msg/Odometry
    /scan0: sensor_msgs/msg/LaserScan
    /scan1: sensor_msgs/msg/LaserScan
  Service Servers:

  Service Clients:

  Action Servers:

  Action Clients:
```

A table is compiled here, showing which topics this node publishes and subscribes to, as well as the message data type for each topic.

Subscription Topic Table

Subscribed Topics	Topic Message Type	Topic Attributes
/arm6_joints	arm_msgs/msg/ArmJoints	Controls six servos
/arm_joint	arm_msgs/msg/ArmJoint	Controls a single servo
/beep	std_msgs/msg/UInt16	Controls the buzzer
/cmd_vel	geometry_msgs/msg/Twist	Controls the car's speed
/rgb	std_msgs/msg/ColorRGBA	Controls wait

Published topic table

Published topic	Topic message type	Topic attributes
/battery	std_msgs/msg/Float32	Publishes battery level data
/imu/data_raw	sensor_msgs/msg/Imu	Publish imu data
/odom_raw	nav_msgs/msg/Odometry	Publish odometry data
/scan0	sensor_msgs/msg/LaserScan	Publish left rear radar data
/rgb	std_msgs/msg/ColorRGBA	Publish right front radar data

<<<<<<< HEAD

3.3 Publishing Control Commands

According to the table of subscribed topics, use the following command format: `ros2 topic pub topic name topic message data type message data --once` to publish a frame of control data.

3.3.1 Controlling the Car's Speed

- Publishing Control Commands

According to the table of subscribed topics, use the following command format: `ros2 topic pub topic name topic message data type message data --once` to publish a frame of control data.

For the first test, it's recommended to set up the car and test it without its wheels touching the ground. Let's set the car to move forward at a linear velocity of 0.1 m/s. Enter the following command in the terminal:

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.1, y: 0.0, z:
0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" --once
```

After running, the car will move forward at a speed of 0.1 m/s. Similarly, to control the car to move at an angular velocity of 1.0 rad/s, assign the z value of angular_r to the following command:

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z:
0.0}} 0.0}, angular: {x: 0.0, y: 0.0, z: 1.0}}" --once
```

After running, the robot will rotate. To stop, simply publish both the linear velocity and angular velocity to 0. The command is as follows:

```
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}}" --once
```

The --once flag indicates that only one frame of message data will be sent. For other parameters for the ros2 topic pub command, please refer to [19. Common ROS2 Command Tools] in [15. ROS2 Basics] of this product course.

3.3.2 Controlling the Car's Buzzer

To turn on the buzzer, enter the following command in the terminal:

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

To turn off the buzzer, enter the following command in the terminal:

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

3.3.3 Controlling the Car's Light Strip

To publish red, enter the following command in the terminal:

```
ros2 topic pub /rgb std_msgs/msg/ColorRGBA "{r: 1.0, g: 0.0, b: 0.0, a: 1.0}" --
once
```

3.3.4 Controlling Six Servos

Set the angles of the six servos to 90 degrees, and the robotic arm to an upward, straight posture. Note: Stand clear of the robotic arm to avoid being hit. Enter the following command in the terminal:

```
ros2 topic pub /arm6_joints arm_msgs/msg/ArmJoints {"joint1: 90, joint2: 90, 
joint3: 90, joint4: 90, joint5: 90, joint6: 90, time: 1500"} --once
```

The time value here represents the servo operation time, in milliseconds.

3.3.5 Controlling a Single Servo

Set the angle of servo #6 (gripper) to 150 degrees, indicating a gripping state. Enter the following command in the terminal:

```
ros2 topic pub /arm_joint arm_msgs/msg/ArmJoint "{id: 6,joint: 150,time: 2000}" 
--once
```

3.4 Subscribing to Car Data

According to the published table, use the command `ros2 topic exho topic-name` in the following format to receive sensor data published by the car node.

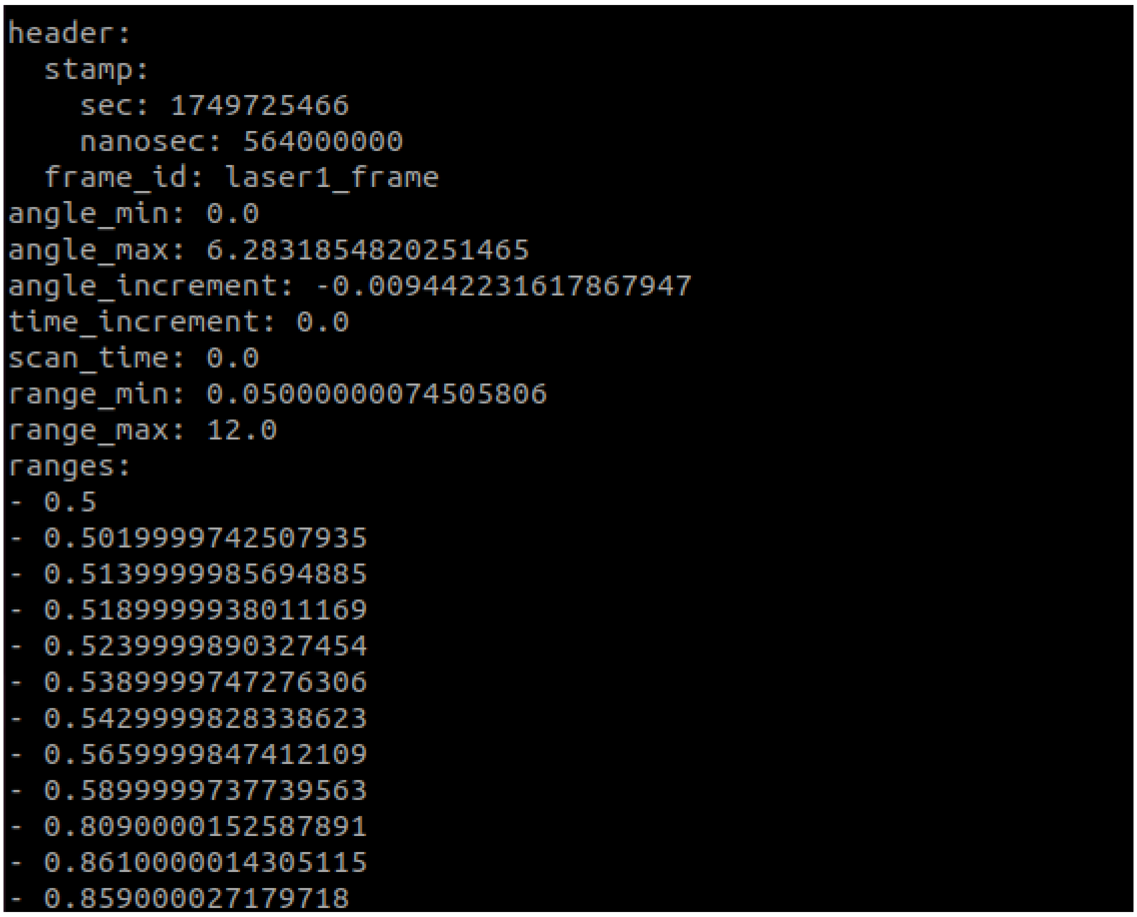
3.4.1 Subscribing to Radar Data

According to the published table, use the command `ros2 topic exho topic-name` in the following format to receive sensor data published by the car node.

This product has two radars. The topic name for the left rear radar is /scan0, and the topic name for the right front radar is /scan1. For example, using the right front radar data, enter the following command in the terminal:

```
ros2 topic echo /scan1
```

The subscribed data is shown in the figure below.



```
header:
  stamp:
    sec: 1749725466
    nanosec: 564000000
  frame_id: laser1_frame
angle_min: 0.0
angle_max: 6.2831854820251465
angle_increment: -0.009442231617867947
time_increment: 0.0
scan_time: 0.0
range_min: 0.05000000074505806
range_max: 12.0
ranges:
- 0.5
- 0.5019999742507935
- 0.5139999985694885
- 0.5189999938011169
- 0.5239999890327454
- 0.5389999747276306
- 0.5429999828338623
- 0.5659999847412109
- 0.5899999737739563
- 0.8090000152587891
- 0.8610000014305115
- 0.859000027179718
```

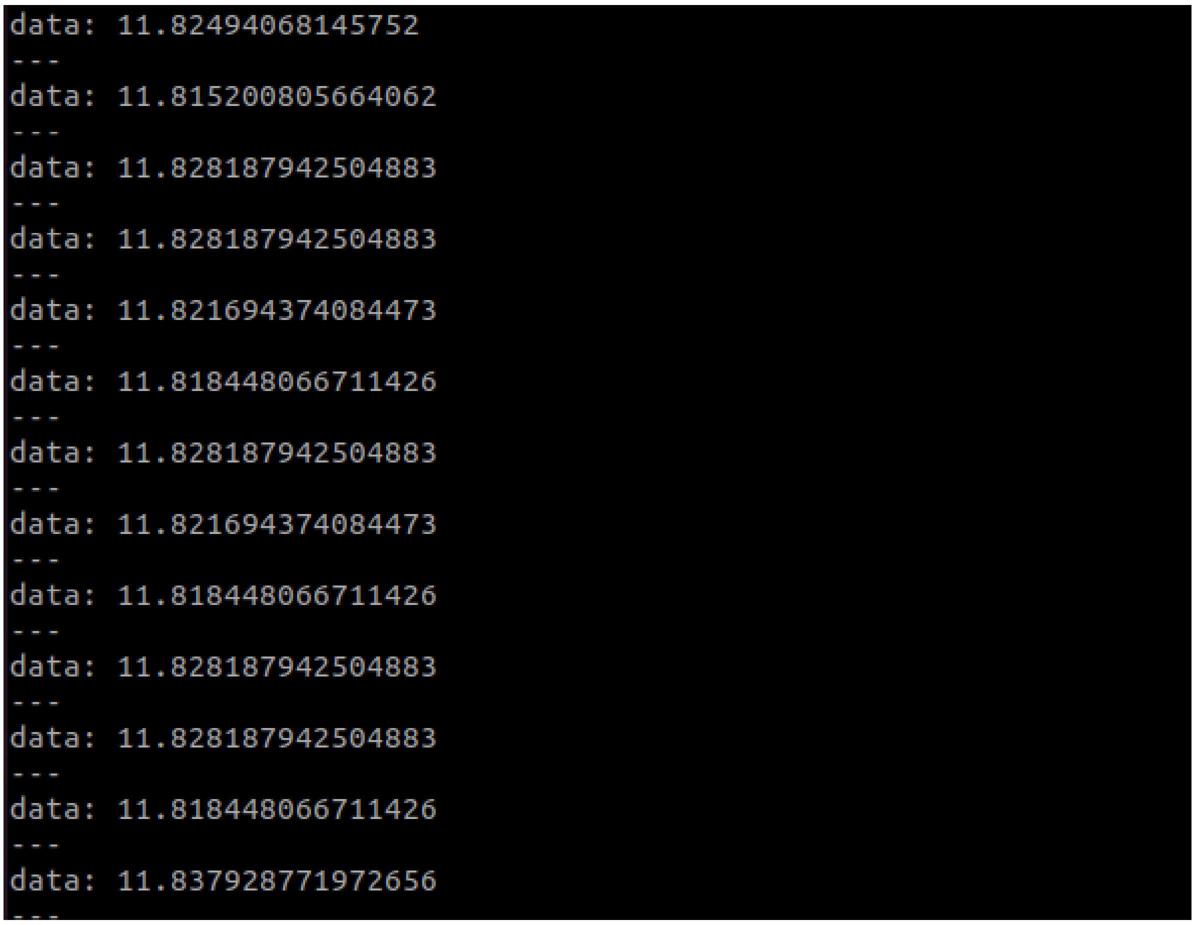
For more information about radar data, please visit [6. LiDAR] in this course.

3.4.2 Subscribing to Battery Level Data

Theoretically, the battery voltage for normal operation of this product should be above 9.6V and below 12V. If it falls below 9.6V, the buzzer will beep, indicating that the battery voltage is too low and needs to be charged. You can enter the following command in the terminal to query the battery voltage:

```
ros2 topic echo /battery
```

The subscribed data is shown in the figure below.


```
data: 11.82494068145752
---
data: 11.815200805664062
---
data: 11.828187942504883
---
data: 11.828187942504883
---
data: 11.821694374084473
---
data: 11.818448066711426
---
data: 11.828187942504883
---
data: 11.821694374084473
---
data: 11.818448066711426
---
data: 11.828187942504883
---
data: 11.828187942504883
---
data: 11.818448066711426
---
data: 11.837928771972656
```

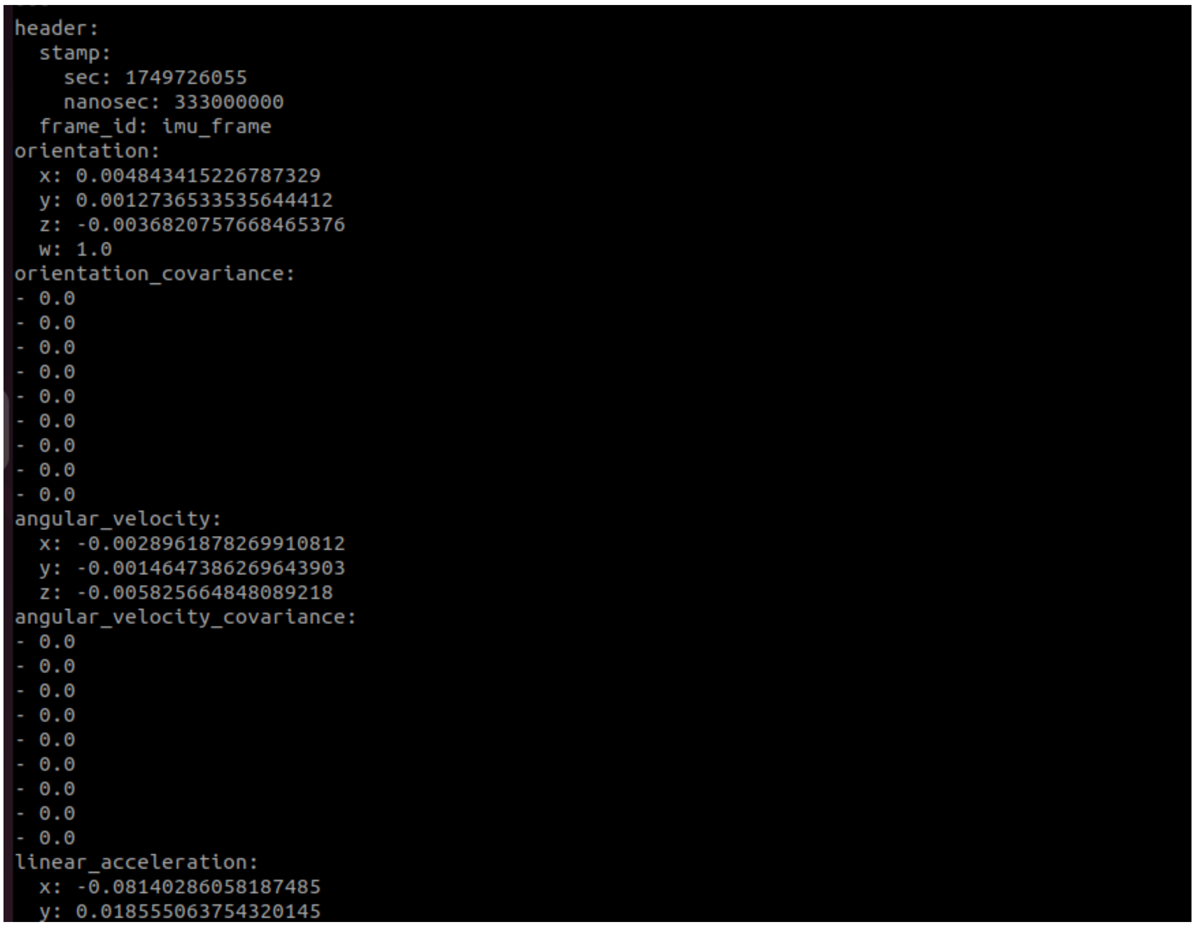
Here, the battery voltage is 11.8V.

3.4.3 Subscribing to IMU Data

The control board has a 9-axis IMU that provides feedback on the car's attitude. Enter the following command in the terminal to retrieve IMU data:

```
ros2 topic echo /imu/data_raw
```

The subscribed IMU data is shown in the figure below.



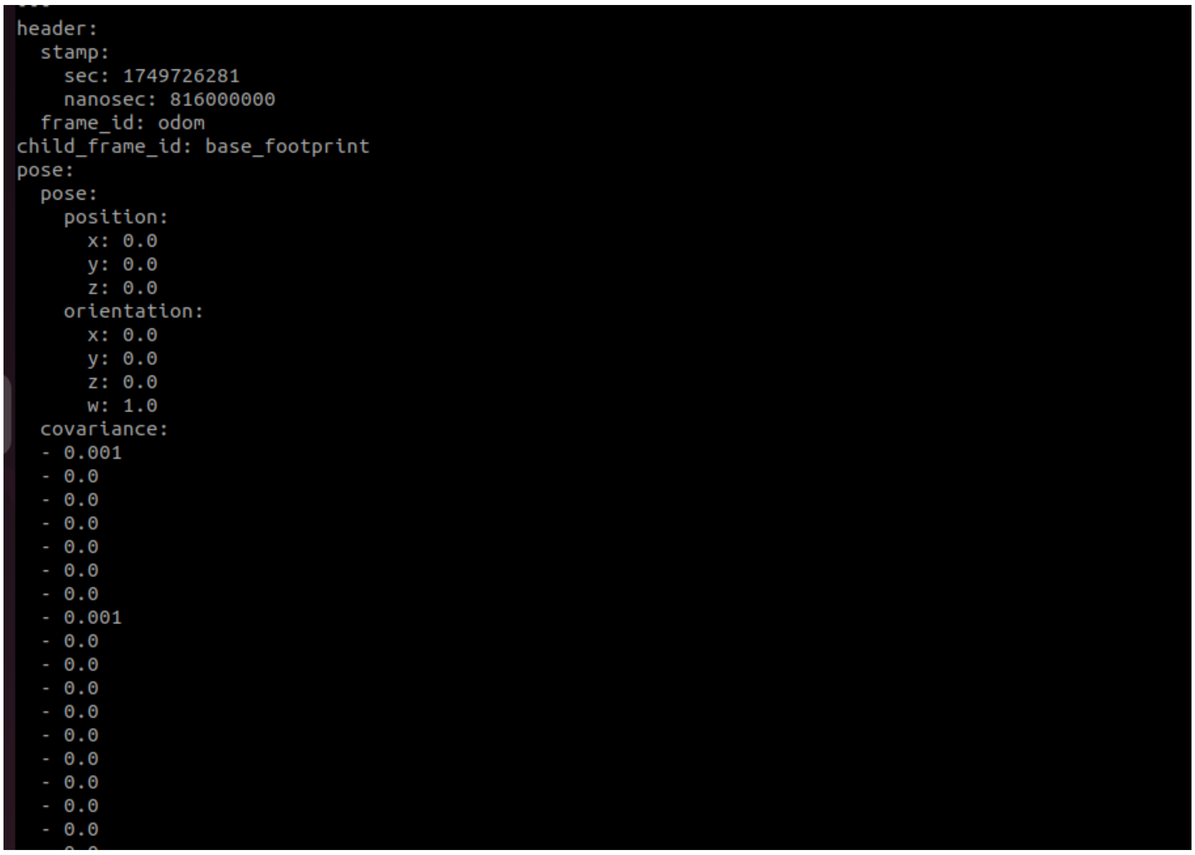
```
header:
  stamp:
    sec: 1749726055
    nanosec: 333000000
  frame_id: imu_frame
orientation:
  x: 0.004843415226787329
  y: 0.0012736533535644412
  z: -0.0036820757668465376
  w: 1.0
orientation_covariance:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
angular_velocity:
  x: -0.0028961878269910812
  y: -0.0014647386269643903
  z: -0.005825664848089218
angular_velocity_covariance:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
linear_acceleration:
  x: -0.08140286058187485
  y: 0.018555063754320145
```

3.4.4 Subscribing to Odometer Data

This product's four motors are equipped with encoders. The ROS control board reads the encoder information and publishes calculated odometer data. Enter the following command in the terminal to read the odometer data:

```
ros2 topic echo /odom_raw
```

Subscribing to odometer data is shown below:



```
header:
  stamp:
    sec: 1749726281
    nanosec: 816000000
  frame_id: odom
child_frame_id: base_footprint
pose:
  pose:
    position:
      x: 0.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
  covariance:
  - 0.001
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.001
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
```