

Multimodal visual understand+SLAM navigation + visual line patrolling(Text Version)

Multimodal visual understand+SLAM navigation + visual line patrolling(Text Version)

1. Course Content
2. Preparation
 - 2.1 Content Description
3. Run Case
 - 3.1 Starting the Program
 - 3.1.1 Jetson Nano board Startup Steps:
 - 3.1.2 Other main control startup steps:
 - 3.2 Test Case
 - 3.2.1 Case 1
4. Source Code Analysis

1. Course Content

1. Learn to use the robot's visual understanding, line following, SLAM navigation, and other complex functional cases.
2. Study the key source code newly introduced in the tutorial.
3. **Note: This section requires you to complete the map file configuration as described in the previous section [7. Multimodal Visual Understand + SLAM Navigation].**

2. Preparation

2.1 Content Description

This lesson uses the Raspberry Pi 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 lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [4.Enter Docker (For JETSON Nano and RPi 5)]**. For RDKX5 and Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

💡 This example uses `model:"qwen/qwen2.5-v1-72b-instruct:free", "qwen-v1-latest"`

⚠️ The responses from the big model for the same test command may not be exactly the same and may differ slightly from the screenshots in the tutorial. To increase or decrease the diversity of the big model's responses, refer to the section on configuring the decision-making big model parameters in **[03.AI Model Basics] -- [5.Configure AI large model]**.

3. Run Case

3.1 Starting the Program

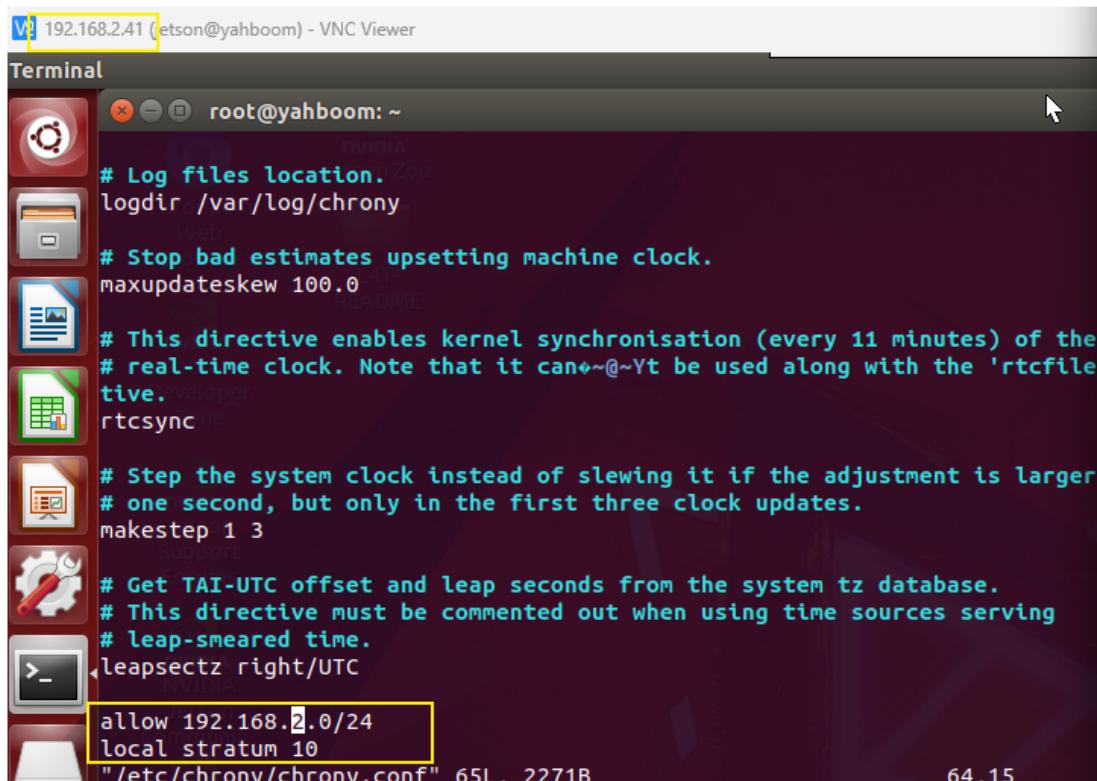
3.1.1 Jetson Nano board Startup Steps:

Due to Nano performance issues, navigation-related nodes must be run on a virtual machine. Therefore, navigation performance is highly dependent on the network. We recommend running in an indoor environment with a good network connection. You must first configure the following:

- **jetson nano (requires entering Docker)**

Open a terminal and enter

```
sudo vi /etc/chrony/chrony.conf
```



Add the following two lines to the end of the file: (Fill in 192.168.2.0/24 based on the actual network segment; this example uses the current board IP address of 192.168.2.41.)

```
allow 192.168.x.0/24 #x indicates the corresponding network segment
local stratum 10
```

After saving and exiting, enter the following command to take effect:

```
sudo service chrony restart
```

```
root@yahboom:~# sudo service chrony restart
 * Stopping time daemon chronyd
 * Starting time daemon chronyd
[ OK ]
[ OK ]
root@yahboom:~#
```

- **Virtual Machine**

Open a terminal and enter

```
echo "server 192.168.2.41 iburst" | sudo tee
/etc/chrony/sources.d/jetson.sources
```

The 192.168.2.41 entry above is the board's IP address.

Enter the following command to take effect.

```
sudo chronyc reload sources
```

Enter the following command again to check the latency. If the IP address appears, it's normal.

```
chronyc sources -v
```

```
yahboom@VM:~$ chronyc sources -v

... Source mode '^' = server, '=' = peer, '#' = local clock.
/.. Source state '*' = current best, '+' = combined, '-' = not combined,
| /          'x' = may be in error, '~' = too variable, '?' = unusable.
||           .- xxxx [ yyyy ] +/- zzzz
||           |   xxxx = adjusted offset,
||           |   yyyy = measured offset,
||           |   zzzz = estimated error.
||           \   |
||           |   |
||           |   \
MS Name/IP address      Stratum Poll Reach LastRx Last sample
=====
^ prod-ntp-5.ntp1.ps5.cano>    2    7    277    133   -4048us[-5395us] +/-  132ms
^- alphyn.canonical.com       2    8    177     66   -193us[-193us] +/-  134ms
^ prod-ntp-3.ntp1.ps5.cano>    2    8    367    131   -16ms[-18ms] +/-  127ms
^ prod-ntp-4.ntp4.ps5.cano>    2    8    377    129   -4907us[-6254us] +/-  133ms
^* time.nju.edu.cn            1    7    377     69   +77us[-1270us] +/-  17ms
^+ 139.199.215.251             2    8    377    135   +2358us[+1011us] +/-  46ms
^+ 119.28.183.184              2    7    377    193   +2061us[+713us] +/-  28ms
^+ 119.28.206.193              2    8    367     8   +2058us[+2058us] +/-  36ms
^+ 192.168.2.41                4    6    377     6   -12ms[-12ms] +/-  92ms
yahboom@VM:~$
```

• Start the Program

On the mainboard, open a terminal in Docker and enter the following command:

```
ros2 launch largemode largemode_control.launch.py
```

```
root@raspbpi:~#
File Edit Tabs Help
root@raspbpi:~ x root@raspbpi:~ x
root@raspbpi:~# ros2 launch largemode largemode_control.launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2025-08-21-16-01-10-206398-raspberrypi-98016
[INFO] [launch]: Default logging verbosity is set to INFO
----- robot_type = A1 -----
----- robot_type = A1 -----
[INFO] [yulidar_ros2_driver_node-10]: process started with pid [98082]
[INFO] [ascamera_node-1]: process started with pid [98050]
[INFO] [joint_state_publisher-2]: process started with pid [98052]
[INFO] [robot_state_publisher-3]: process started with pid [98054]
[INFO] [Ackman_driver_A1-4]: process started with pid [98056]
[INFO] [base_node_A1-5]: process started with pid [98058]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [98061]
[INFO] [ekf_node-7]: process started with pid [98065]
[INFO] [yahboom_joy_A1-8]: process started with pid [98075]
[INFO] [joy_node-9]: process started with pid [98078]
[INFO] [static_transform_publisher-11]: process started with pid [98090]
[INFO] [static_transform_publisher-12]: process started with pid [98092]
[INFO] [static_transform_publisher-13]: process started with pid [98095]
[INFO] [model_service-14]: process started with pid [98100]
[INFO] [action_service_nuwa-15]: process started with pid [98102]
[INFO] [asr-16]: process started with pid [98105]
[robot_state_publisher-3] [INFO] [1755763271.586916300] [robot_state_publisher]: got segment base_footprint
[robot_state_publisher-3] [INFO] [1755763271.587125040] [robot_state_publisher]: got segment base_link
[robot_state_publisher-3] [INFO] [1755763271.587149058] [robot_state_publisher]: got segment camera_link
[robot_state_publisher-3] [INFO] [1755763271.587160632] [robot_state_publisher]: got segment imu_link
[robot_state_publisher-3] [INFO] [1755763271.587171336] [robot_state_publisher]: got segment laser
[robot_state_publisher-3] [INFO] [1755763271.587180410] [robot_state_publisher]: got segment left_front_wheel_joint
[robot_state_publisher-3] [INFO] [1755763271.587180632] [robot_state_publisher]: got segment left_rear_wheel_hinge
[robot_state_publisher-3] [INFO] [1755763271.587199428] [robot_state_publisher]: got segment left_steering_hinge_joint
[robot_state_publisher-3] [INFO] [1755763271.587208447] [robot_state_publisher]: got segment right_front_wheel_joint
[robot_state_publisher-3] [INFO] [1755763271.587217132] [robot_state_publisher]: got segment right_rear_wheel_hinge
[robot_state_publisher-3] [INFO] [1755763271.587226428] [robot_state_publisher]: got segment right_steering_hinge_joint
```

On the virtual machine, create a new terminal and start.

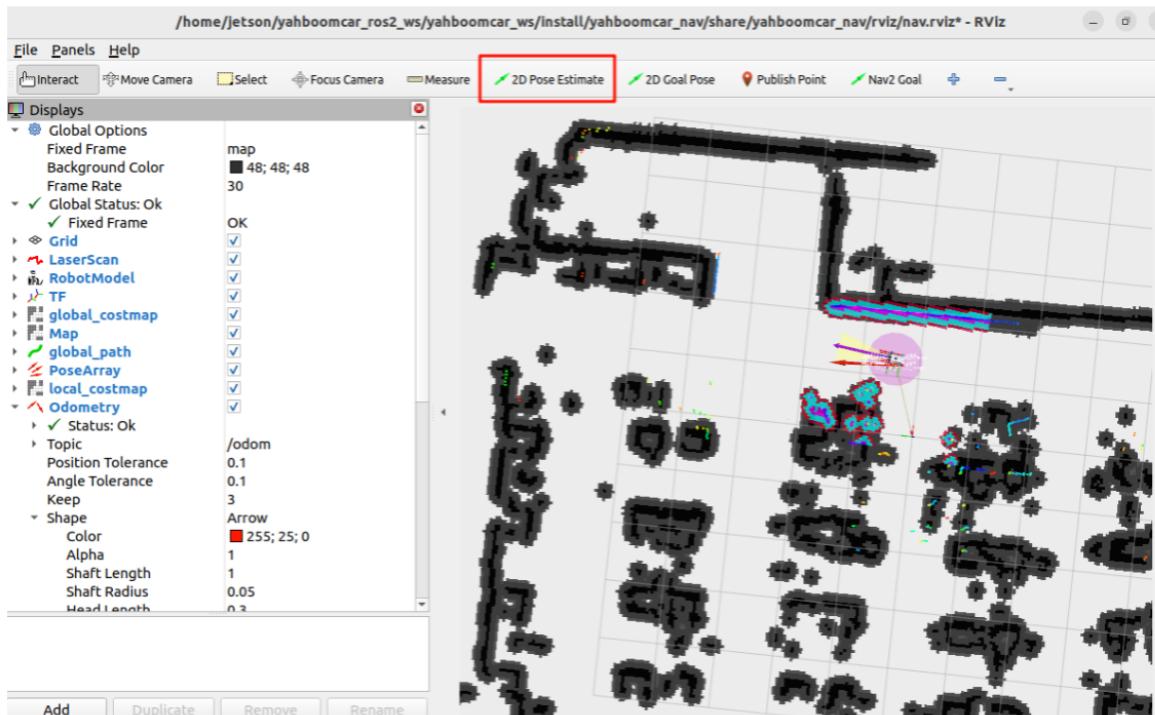
```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start before displaying the map. Then open a VM terminal and enter:

```
ros2 launch yahboomcar_nav navigation_teb_launch.py
```

Note: When running the car's mapping function, you must enter the save map command on the VM to save the navigation map.

After that, follow the navigation function startup process to initialize positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to select it. Roughly mark the robot's position and orientation on the map. After initializing positioning, preparations are complete.



3.1.2 Other main control startup steps:

For the Raspberry Pi 5, you need to first enter the Docker container; the ORIN board does not require this.

Open a terminal in Docker and enter the following command:

```
ros2 launch largemode1 largemode1_control.launch.py text_chat_mode:=True
```

```

root@raspb... ~ root@raspb... ~ root@raspb... ~ root@raspb... ~
root@raspb...# ros2 launch largemode1 largemode1.launch.py text_chat_mode:=True
[INFO] [launch]: All log files can be found below /root/.ros/log/2025-08-20-15-20-59-679615-raspberrypi-87377
[INFO] [launch]: Default logging verbosity is set to INFO
----- robot_type = A1, rplidar_type = tmini, camera_type = nuwa -----
-----robot_type = A1-----
[INFO] [ydlidar_ros2_driver_node-10]: process started with pid [87437]
[INFO] [ascamera_node-1]: process started with pid [87409]
[INFO] [joint_state_publisher-2]: process started with pid [87411]
[INFO] [robot_state_publisher-3]: process started with pid [87413]
[INFO] [Ackman_driver_A1-4]: process started with pid [87415]
[INFO] [base_node_A1-5]: process started with pid [87417]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [87420]
[INFO] [ekf_node-7]: process started with pid [87425]
[INFO] [yahboom_joy_A1-8]: process started with pid [87432]
[INFO] [joy_node-9]: process started with pid [87435]
[INFO] [static_transform_publisher-11]: process started with pid [87440]
[INFO] [static_transform_publisher-12]: process started with pid [87449]
[INFO] [static_transform_publisher-13]: process started with pid [87451]
[INFO] [model_service-14]: process started with pid [87454]
[INFO] [action_service_nuwa-15]: process started with pid [87457]
[ascamera_node-1] [INFO] [1755674461.008407074] [ascamera_hp60c.camera_publisher]: get depth_width 640
[ascamera_node-1] [INFO] [1755674461.008622721] [ascamera_hp60c.camera_publisher]: get depth_height 480
[ascamera_node-1] [INFO] [1755674461.008647609] [ascamera_hp60c.camera_publisher]: get rgb_width 640
[ascamera_node-1] [INFO] [1755674461.008662739] [ascamera_hp60c.camera_publisher]: get rgb_height 480
[ascamera_node-1] [INFO] [1755674461.008674739] [ascamera_hp60c.camera_publisher]: get set_fps 25
[ascamera_node-1] [INFO] [1755674461.008685739] [ascamera_hp60c.camera_publisher]: get pub_tfTree 1
[ascamera_node-1] [INFO] [1755674461.008696609] [ascamera_hp60c.camera_publisher]: hello world angstrong camera ros2 node
[ascamera_node-1] [INFO] [1755674461.008797201] [ascamera_hp60c.camera_publisher]: 2025-08-20 15:21:01[INFO] [CameraSrv.cpp] [35] [CameraSrv] Angstrong camera server
[ascamera_node-1] [INFO] [1755674461.008831553] [ascamera_hp60c.camera_publisher]: 2025-08-20 15:21:01[INFO] [CameraSrv.cpp] [45] [CameraSrv] Angstrong camera
sdk version:v1.2.28.20241021
[static_transform_publisher-13] [WARN] [1755674461.049432038] []: Old-style arguments are deprecated; see --help for new-style arguments

```

Create a new terminal on the virtual machine and start the command.** (For Raspberry Pi and Jetson Nano, it is recommended to run the visualization in the virtual machine)**

```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start and the image will be displayed. Then open the Docker terminal and enter the following command:

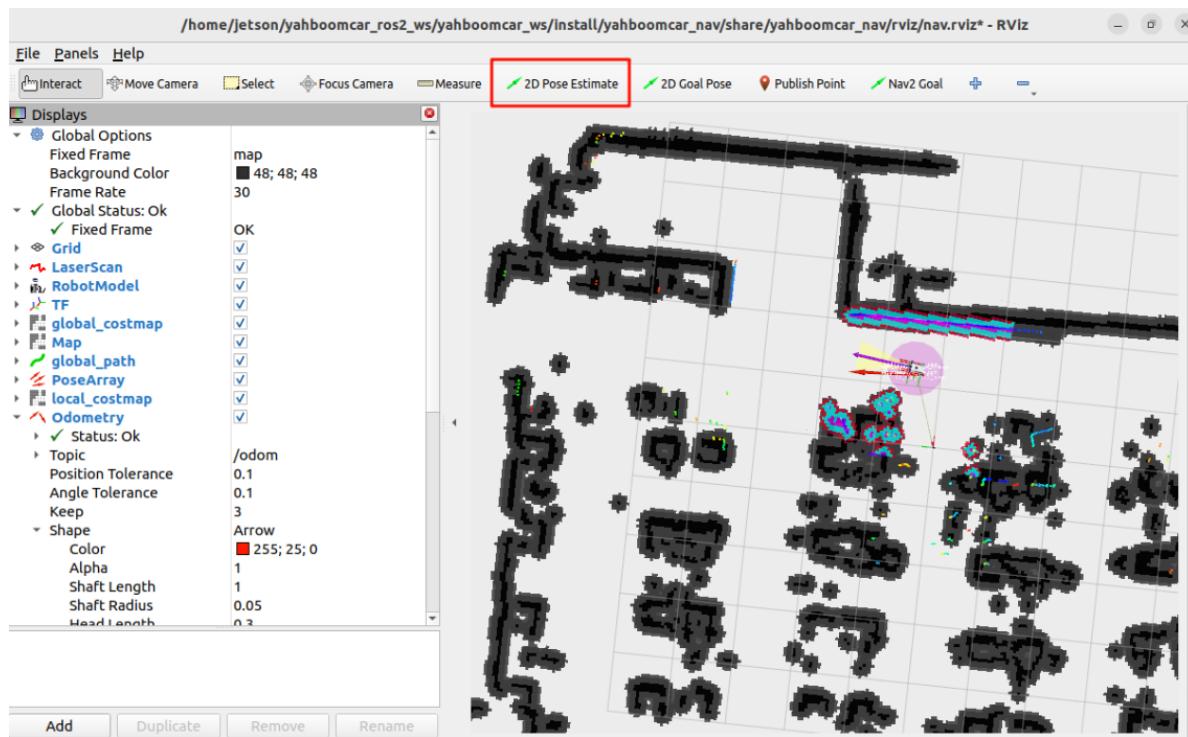
```
#Choose one of the two navigation algorithms
#Standard navigation
ros2 launch yahboomcar_nav navigation_teb_launch.py

#Fast relocalization navigation(RDKX5, Raspberry Pi 5, and Jetson Nano are not
#supported)
ros2 launch yahboomcar_nav localization_imu_odom.launch.py use_rviz:=false
load_state_filename:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomca
r_nav/maps/yahboomcar.pbstream

ros2 launch yahboomcar_nav navigation_cartodwb_launch.py
maps:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahb
oomcar.yaml
params_file:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/pa
rams/cartoteb_nav_params.yaml
```

Note: yahboomcar.yaml and yahboomcar.pbstream must be mapped simultaneously, meaning they are the same map. Refer to the cartograph mapping algorithm for saving maps.

After that, follow the navigation function startup process to initialize positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to enter the selection state. Roughly mark the robot's position and orientation on the map. After initial positioning, preparations are complete.



Open the same docker in multiple terminals and start it.

```
ros2 run text_chat text_chat
```

```
root@raspberrypi:~# ros2 run text_chat text_chat
user input: [
```

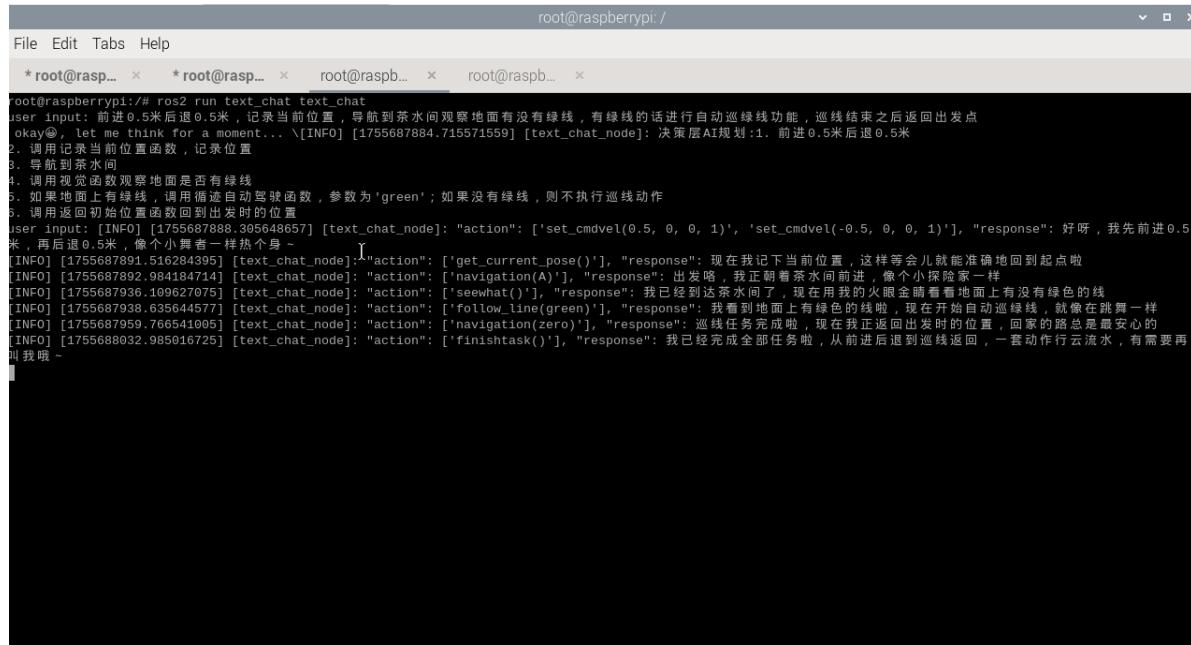
3.2 Test Case

Here are some reference test cases; users can create their own dialogue commands.

- Move forward 0.5 meters and back 0.5 meters, record the current location, navigate to the tea room, and observe if there's a green line on the floor. If there is, automatically patrol the green line. After patrolling, return to the starting point.

3.2.1 Case 1

In the terminal, enter "Move forward 0.5 meters and back 0.5 meters, record the current location, navigate to the tea room, and observe if there's a green line on the floor. If there is, automatically patrol the green line. After patrolling, return to the starting point." The terminal prints the following information.



The terminal window shows a root shell on a Raspberry Pi. It displays the following sequence of events:

- User input: 前进0.5米后退0.5米，记录当前位置，导航到茶水间观察地面是否有绿线，有绿线的话进行自动巡绿线功能，巡线结束之后返回出发点
- 决策层AI规划:1. 前进0.5米后退0.5米
- 调用记录当前位置函数，记录位置
- 导航到茶水间
- 调用视觉函数观察地面是否有绿线
- 如果地面上有绿线，调用循迹自动驾驶函数，参数为'green'；如果没有绿线，则不执行巡线动作
- User input: [INFO] [1755687888.365648657] [text_chat_node]: "action": ["set_cmdvel(0.5, 0, 0, 1)", "set_cmdvel(-0.5, 0, 0, 1)"], "response": 好呀，我先前进0.5米，再后退0.5米，像个小舞者一样热个身~
- [INFO] [1755687891.516284395] [text_chat_node]: "action": ["get_current_pose()"], "response": 现在我记下当前位置，这样等会儿就能准确地回到起点啦
- [INFO] [1755687892.984184714] [text_chat_node]: "action": ["navigation(A)"], "response": 出发咯，我正朝着茶水间前进，像个小探险家一样
- [INFO] [1755687936.109627075] [text_chat_node]: "action": ["seewhat()"], "response": 我已经到达茶水间了，现在用我的火眼金睛看看地面上有没有绿色的线
- [INFO] [1755687938.635644577] [text_chat_node]: "action": ["follow_line(green)"], "response": 我看到地面上有绿色的线啦，现在开始自动巡绿线，就像在跳舞一样
- [INFO] [1755687959.766541065] [text_chat_node]: "action": ["navigation(zero)"], "response": 巡线任务完成啦，现在我正返回出发时的位置，回家的路总是最安心的
- [INFO] [1755688032.985016725] [text_chat_node]: "action": ["finishtask()"], "response": 我已经完成全部任务啦，从前进后退到巡线返回，一套动作行云流水，有需要再叫我哦~

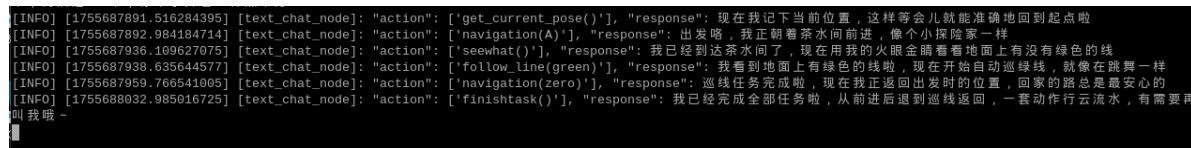
The decision-making layer model outputs the planned task steps:



The terminal window shows the planned task steps from the decision-making layer:

2. 调用记录当前位置函数，记录位置
3. 导航到茶水间
4. 调用视觉函数观察地面是否有绿线
5. 如果地面上有绿线，调用循迹自动驾驶函数，参数为'green'；如果没有绿线，则不执行巡线动作
6. 调用返回初始位置函数回到出发时的位置

The execution layer model will then execute the task steps:



The terminal window shows the execution of the task steps from the execution layer:

- [INFO] [1755687891.516284395] [text_chat_node]: "action": ["get_current_pose()"], "response": 现在我记下当前位置，这样等会儿就能准确地回到起点啦
- [INFO] [1755687892.984184714] [text_chat_node]: "action": ["navigation(A)"], "response": 出发咯，我正朝着茶水间前进，像个小探险家一样
- [INFO] [1755687936.109627075] [text_chat_node]: "action": ["seewhat()"], "response": 我已经到达茶水间了，现在用我的火眼金睛看看地面上有没有绿色的线
- [INFO] [1755687938.635644577] [text_chat_node]: "action": ["follow_line(green)"], "response": 我看到地面上有绿色的线啦，现在开始自动巡绿线，就像在跳舞一样
- [INFO] [1755687959.766541065] [text_chat_node]: "action": ["navigation(zero)"], "response": 巡线任务完成啦，现在我正返回出发时的位置，回家的路总是最安心的
- [INFO] [1755688032.985016725] [text_chat_node]: "action": ["finishtask()"], "response": 我已经完成全部任务啦，从前进后退到巡线返回，一套动作行云流水，有需要再叫我哦~

After completing the task, the robot will enter a waiting state. During this time, commands are directly passed to the execution layer model, and all conversation history is retained. You can press the `ENTER` key in the terminal to continue the conversation and enter the "End current task" command to terminate the current task cycle and start a new one.

```

root@raspberrypi:/# ros2 run text_chat text_chat
user input: 前进0.5米后退0.5米，记录当前位置，导航到茶水间观察地面是否有绿线，有绿线的话进行自动巡绿线功能，巡线结束之后返回出发点
okay@, let me think for a moment... \[INFO] [1755687884.715571559] [text_chat_node]: 决策层AI规划:1. 前进0.5米后退0.5米
2. 调用记录当前位置函数，记录位置
3. 导航到茶水间
4. 调用视觉函数观察地面是否有绿线
5. 如果地面上有绿线，调用循迹自动驾驶函数，参数为'green'；如果没有绿线，则不执行巡线动作
6. 调用返回初始位置函数回到出发时的位置
user input: \[INFO] [1755687888.305648657] [text_chat_node]: "action": ["set_cmdvel(0.5, 0, 0, 1)", "set_cmdvel(-0.5, 0, 0, 1)"], "response": 好呀，我先前进0.5米，再后退0.5米，像个小舞者一样热个身～
\[INFO] [1755687891.516284395] [text_chat_node]: "action": ["get_current_pose()"], "response": 现在我记下当前位置，这样等会儿就能准确地回到起点啦
\[INFO] [1755687892.084184744] [text_chat_node]: "action": ["navigation(A)"], "response": 出发咯，我正朝着茶水间前进，像个小探险家一样
\[INFO] [1755687936.109627075] [text_chat_node]: "action": ["seewhat()"], "response": 我已经到达茶水间了，现在用我的火眼金睛看看地面上有没有绿色的线
\[INFO] [1755687938.635644577] [text_chat_node]: "action": ["follow_line(green)"], "response": 我看到地面上有绿色的线啦，现在开始自动巡绿线，就像在跳舞一样
\[INFO] [1755687959.766541005] [text_chat_node]: "action": ["navigation(zero)"], "response": 巡线任务完成啦，现在我正返回出发时的位置，回家的路总是最安心的
\[INFO] [1755688032.985016725] [text_chat_node]: "action": ["finishtask()"], "response": 我已完成全部任务啦，从前进后退到巡线返回，一套动作行云流水，有需要再叫我哦～
user input: 结束当前任务
okay@, let me think for a moment... \[INFO] [1755688265.628314298] [text_chat_node]: "action": ["finish_dialogue()"], "response": 好的，任务已经结束了，有需要再叫我哦～
user input: 

```

4. Source Code Analysis

Source code location:

Jetson Orin Nano host:

```

#NUWA camera user
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_nuwa.py
#USB camera user
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_usb.py

```

RDK X5:

```

#NUWA camera user
/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_nuwa.py
#USB camera user
/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_usb.py

```

Jetson Nano, Raspberry Pi host:

You need to first enter Docker.

```

#NUWA Camera User
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_nuwa.py
#USB Camera User
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_usb.py

```

This example uses the **seewhat**, **navigation**, **load_target_points**, **get_current_pose**, and **follow_line(self, color)** methods from the **CustomActionServer** class. Most of these methods have been explained in the **Multimodal Visual Understanding + SLAM Navigation** section. For a detailed explanation of the **follow_line(self, color)** function, refer to the **Multimodal Visual**

Understanding + Visual Line Patrolling section.