Multimodal visual understand + visual line patroling(Text Version)

1. Course Content

1. Learn to use the robot's visual understanding capabilities in conjunction with line-following autonomous driving.
2. Analyze newly discovered key source code.

2. Preparation

2.1 Content Description

> This course uses the Jetson Orin NANO as an example. This example uses the gimbal servo USB camera version. 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 **[01. Robot Configuration and Operation Guide] -- [5.Enter Docker (For JETSON Nano and RPi 5)]**. For Orin boards, simply open a terminal and enter the commands mentioned in this course.

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

⚠ The responses from the large 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 large model's responses, refer to the section on configuring the decision-making large model parameters in the **[03.AI Model Basics] -- [5.Configure AI large model]** .

⚡ It is recommended that you first try the previous visual example. This example adds voice functionality to the singleton example, and while the functionality is largely the same, I will not elaborate on the implementation process, code debugging, or results.

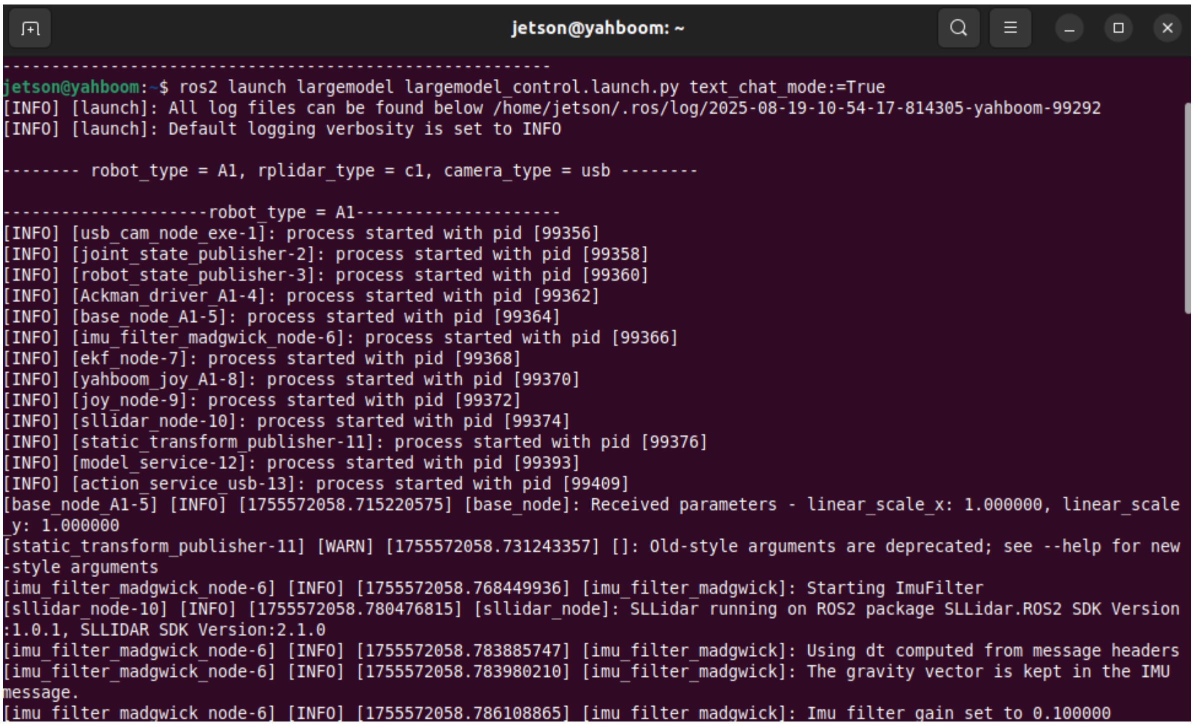
3. Running the Example

3.1 Starting the Program

For Raspberry Pi 5 and Jetson Nano controllers, you must first enter the Docker container. This is not necessary for the Orin board.

Open a terminal on the vehicle and enter the following command:

```
ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
```



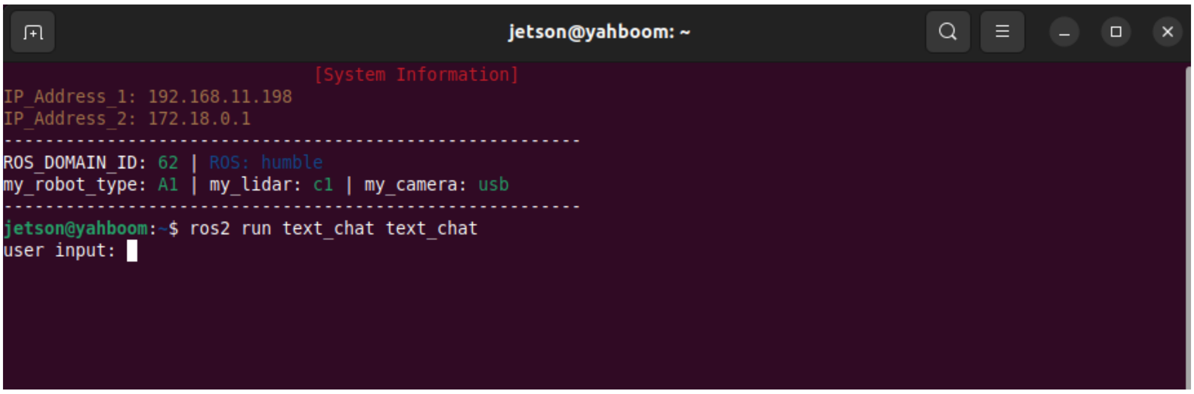
```
--------------------------------------------------------
jetson@yahboom:~$ ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
[INFO] [launch]: All log files can be found below /home/jetson/.ros/log/2025-08-19-10-54-17-814305-yahboom-99292
[INFO] [launch]: Default logging verbosity is set to INFO

-------- robot_type = A1, rplidar_type = c1, camera_type = usb --------

---------------------robot_type = A1---------------------
[INFO] [usb_cam_node_exe-1]: process started with pid [99356]
[INFO] [joint_state_publisher-2]: process started with pid [99358]
[INFO] [robot_state_publisher-3]: process started with pid [99360]
[INFO] [Ackman_driver_A1-4]: process started with pid [99362]
[INFO] [base_node_A1-5]: process started with pid [99364]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [99366]
[INFO] [ekf_node-7]: process started with pid [99368]
[INFO] [yahboom_joy_A1-8]: process started with pid [99370]
[INFO] [joy_node-9]: process started with pid [99372]
[INFO] [sllidar_node-10]: process started with pid [99374]
[INFO] [static_transform_publisher-11]: process started with pid [99376]
[INFO] [model_service-12]: process started with pid [99393]
[INFO] [action_service_usb-13]: process started with pid [99409]
[base_node_A1-5] [INFO] [1755572058.715220575] [base_node]: Received parameters - linear_scale_x: 1.000000, linear_scale
_y: 1.000000
[static_transform_publisher-11] [WARN] [1755572058.731243357] []: Old-style arguments are deprecated; see --help for new
-style arguments
[imu_filter_madgwick_node-6] [INFO] [1755572058.768449936] [imu_filter_madgwick]: Starting ImuFilter
[sllidar_node-10] [INFO] [1755572058.780476815] [sllidar_node]: SLLidar running on ROS2 package SLLidar.ROS2 SDK Version
:1.0.1, SLLIDAR SDK Version:2.1.0
[imu_filter_madgwick_node-6] [INFO] [1755572058.783885747] [imu_filter_madgwick]: Using dt computed from message headers
[imu_filter_madgwick_node-6] [INFO] [1755572058.783980210] [imu_filter_madgwick]: The gravity vector is kept in the IMU
message.
[imu_filter_madgwick_node-6] [INFO] [1755572058.786108865] [imu_filter_madgwick]: Imu filter gain set to 0.100000
```

Open another terminal and start the program:

```
ros2 run text_chat text_chat
```



```
                               [System Information]
IP_Address_1: 192.168.11.198
IP_Address_2: 172.18.0.1
-----------------------------------------------------------
ROS_DOMAIN_ID: 62 | ROS: humble
my_robot_type: A1 | my_lidar: c1 | my_camera: usb
-----------------------------------------------------------
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: 
```

3.2 Test Cases

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

- Start Color Line Patrol

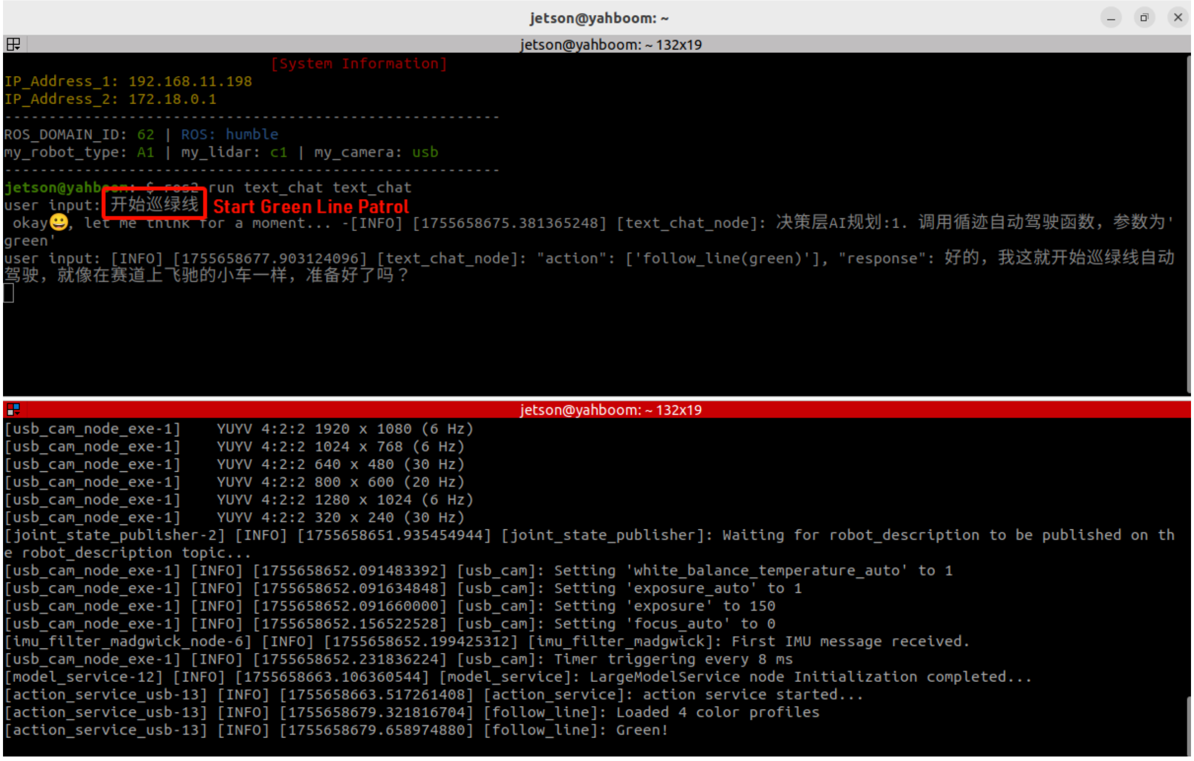
Colors include: red, green, blue, and yellow. (Color calibration is required according to the **AI Large Model Preparation** tutorial.)

⚠ Please do not include a period or any other characters at the end of the text you enter!

3.2.1 Example: "Start Green Line Patrol"

- For users of the gimbal, servo, and USB camera versions, when the program starts, the gimbal and servo will be facing downward by default.
- For users of the depth camera version, you need to adjust the camera angle appropriately.

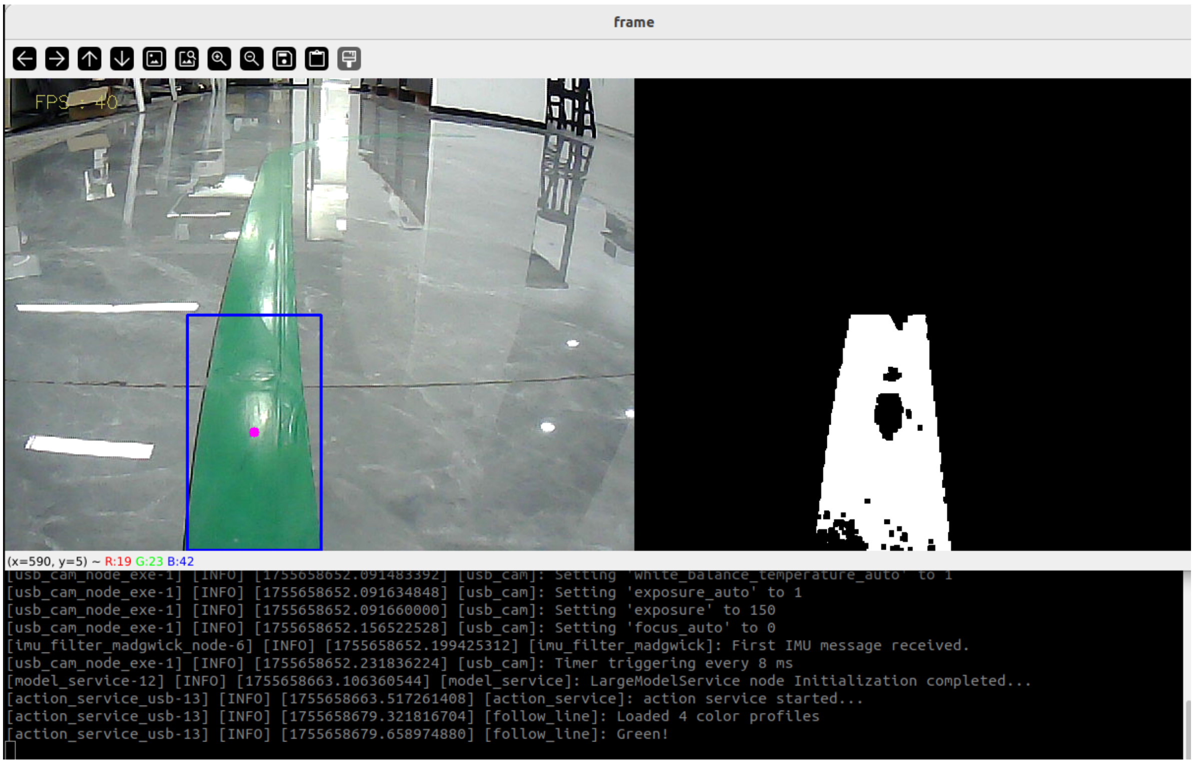
Enter "Start Green Line Patrol" in the terminal. The terminal will print the following information:



```
                                                     [System Information]
IP_Address_1: 192.168.11.198
IP_Address_2: 172.18.0.1
-------------------------------------------------------------------
ROS_DOMAIN_ID: 62 | ROS: humble
my_robot_type: A1 | my_lidar: c1 | my_camera: usb
-------------------------------------------------------------------
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: 开始巡绿线 Start Green Line Patrol
 okay😀, let me think for a moment... -[INFO] [1755658675.381365248] [text_chat_node]: 决策层AI规划:1. 调用循迹自动驾驶函数，参数为'
green'
user input: [INFO] [1755658677.903124096] [text_chat_node]: "action": ['follow_line(green)'], "response": 好的，我这就开始巡绿线自动
驾驶，就像在赛道上飞驰的小车一样，准备好了吗？
```

```
[usb_cam_node_exe-1]    YUYV 4:2:2 1920 x 1080 (6 Hz)
[usb_cam_node_exe-1]    YUYV 4:2:2 1024 x 768 (6 Hz)
[usb_cam_node_exe-1]    YUYV 4:2:2 640 x 480 (30 Hz)
[usb_cam_node_exe-1]    YUYV 4:2:2 800 x 600 (20 Hz)
[usb_cam_node_exe-1]    YUYV 4:2:2 1280 x 1024 (6 Hz)
[usb_cam_node_exe-1]    YUYV 4:2:2 320 x 240 (30 Hz)
[joint_state_publisher-2] [INFO] [1755658651.935454944] [joint_state_publisher]: Waiting for robot_description to be published on th
e robot_description topic...
[usb_cam_node_exe-1] [INFO] [1755658652.091483392] [usb_cam]: Setting 'white_balance_temperature_auto' to 1
[usb_cam_node_exe-1] [INFO] [1755658652.091634848] [usb_cam]: Setting 'exposure_auto' to 1
[usb_cam_node_exe-1] [INFO] [1755658652.091660000] [usb_cam]: Setting 'exposure' to 150
[usb_cam_node_exe-1] [INFO] [1755658652.156522528] [usb_cam]: Setting 'focus_auto' to 0
[imu_filter_madgwick_node-6] [INFO] [1755658652.199425312] [imu_filter_madgwick]: First IMU message received.
[usb_cam_node_exe-1] [INFO] [1755658652.231836224] [usb_cam]: Timer triggering every 8 ms
[model_service-12] [INFO] [1755658663.106360544] [model_service]: LargeModelService node Initialization completed...
[action_service_usb-13] [INFO] [1755658663.517261408] [action_service]: action service started...
[action_service_usb-13] [INFO] [1755658679.321816704] [follow_line]: Loaded 4 color profiles
[action_service_usb-13] [INFO] [1755658679.658974880] [follow_line]: Green!
```

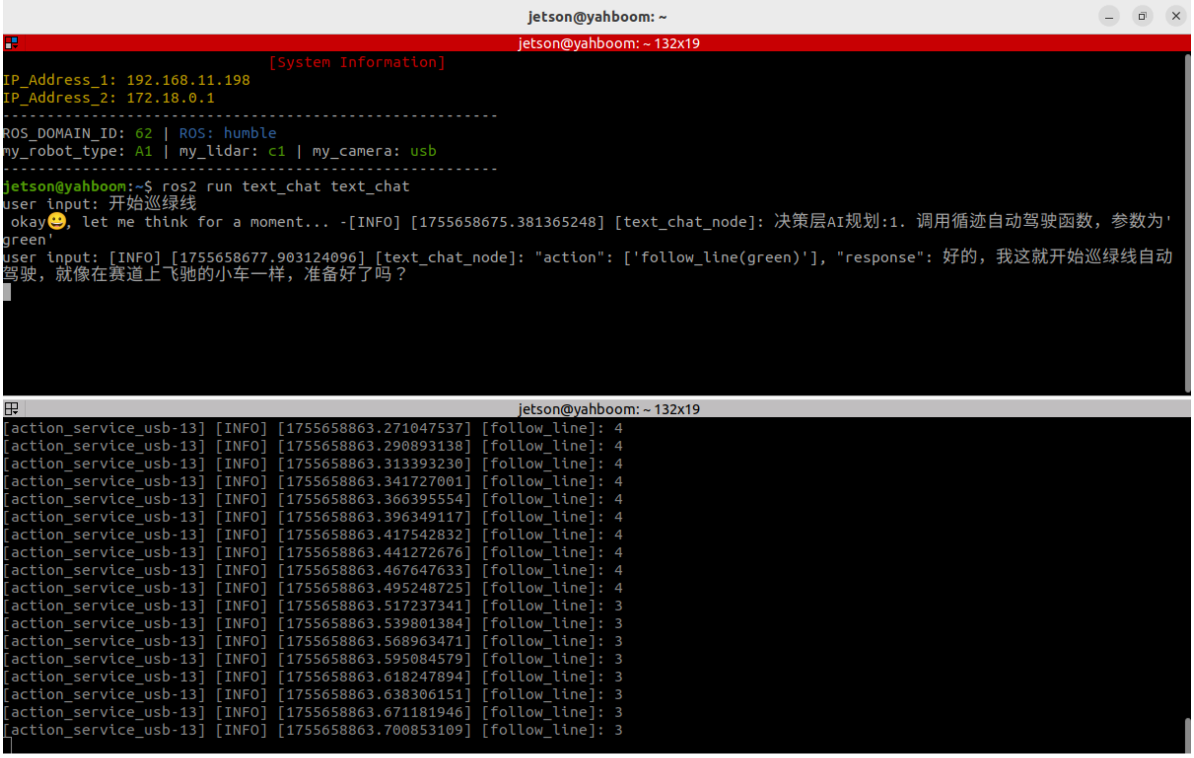
A window titled **frame** will open on the VNC screen, displaying the image from the current robot's perspective.



```
[usb_cam_node_exe-1] [INFO] [1755658652.091483392] [usb_cam]: Setting 'white_balance_temperature_auto' to 1
[usb_cam_node_exe-1] [INFO] [1755658652.091634848] [usb_cam]: Setting 'exposure_auto' to 1
[usb_cam_node_exe-1] [INFO] [1755658652.091660000] [usb_cam]: Setting 'exposure' to 150
[usb_cam_node_exe-1] [INFO] [1755658652.156522528] [usb_cam]: Setting 'focus_auto' to 0
[imu_filter_madgwick_node-6] [INFO] [1755658652.199425312] [imu_filter_madgwick]: First IMU message received.
[usb_cam_node_exe-1] [INFO] [1755658652.231836224] [usb_cam]: Timer triggering every 8 ms
[model_service-12] [INFO] [1755658663.106360544] [model_service]: LargeModelService node Initialization completed...
[action_service_usb-13] [INFO] [1755658663.517261408] [action_service]: action service started...
[action_service_usb-13] [INFO] [1755658679.321816704] [follow_line]: Loaded 4 color profiles
[action_service_usb-13] [INFO] [1755658679.658974880] [follow_line]: Green!
```

Then, the robot will begin calculating its speed and automatically patrol the line. If it encounters an obstacle during patrol, it will stop and the buzzer will sound.

If there are no targets to track in the image, the program will count down for 10 seconds, a 5-second countdown will be printed on the terminal, and the process will automatically end, indicating that the task is complete.

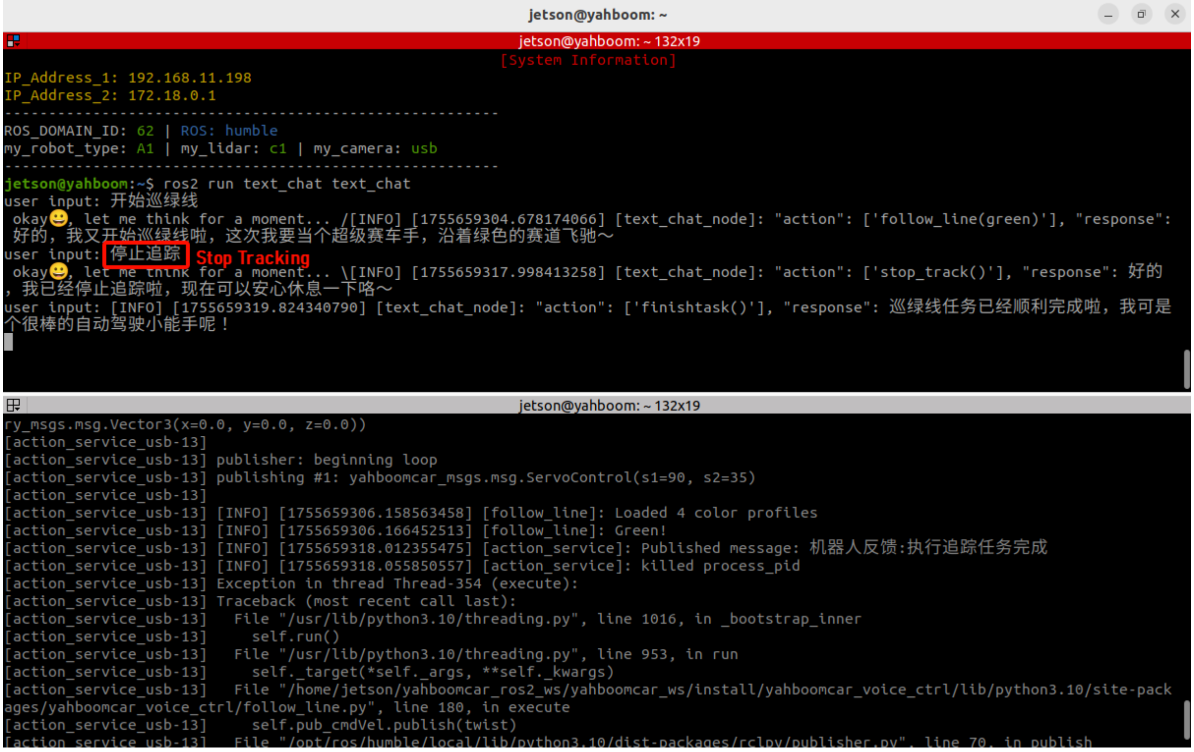


```
                                              [System Information]
IP_Address_1: 192.168.11.198
IP_Address_2: 172.18.0.1
-------------------------------------------------------
ROS_DOMAIN_ID: 62 | ROS: humble
my_robot_type: A1 | my_lidar: c1 | my_camera: usb
-------------------------------------------------------
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: 开始巡绿线
 okay😀, let me think for a moment... -[INFO] [1755658675.381365248] [text_chat_node]: 决策层AI规划:1. 调用循迹自动驾驶函数，参数为'
green'
user input: [INFO] [1755658677.903124096] [text_chat_node]: "action": ['follow_line(green)'], "response": 好的，我这就开始巡绿线自动
驾驶，就像在赛道上飞驰的小车一样，准备好了吗？
```

```
[action_service_usb-13] [INFO] [1755658863.271047537] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.290893138] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.313393230] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.341727001] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.366395554] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.396349117] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.417542832] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.441272676] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.467647633] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.495248725] [follow_line]: 4
[action_service_usb-13] [INFO] [1755658863.517237341] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.539801384] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.568963471] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.595084579] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.618247894] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.638306151] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.671181946] [follow_line]: 3
[action_service_usb-13] [INFO] [1755658863.700853109] [follow_line]: 3
```

To manually end the task, press the `ENTER` key in the terminal and continue the conversation.

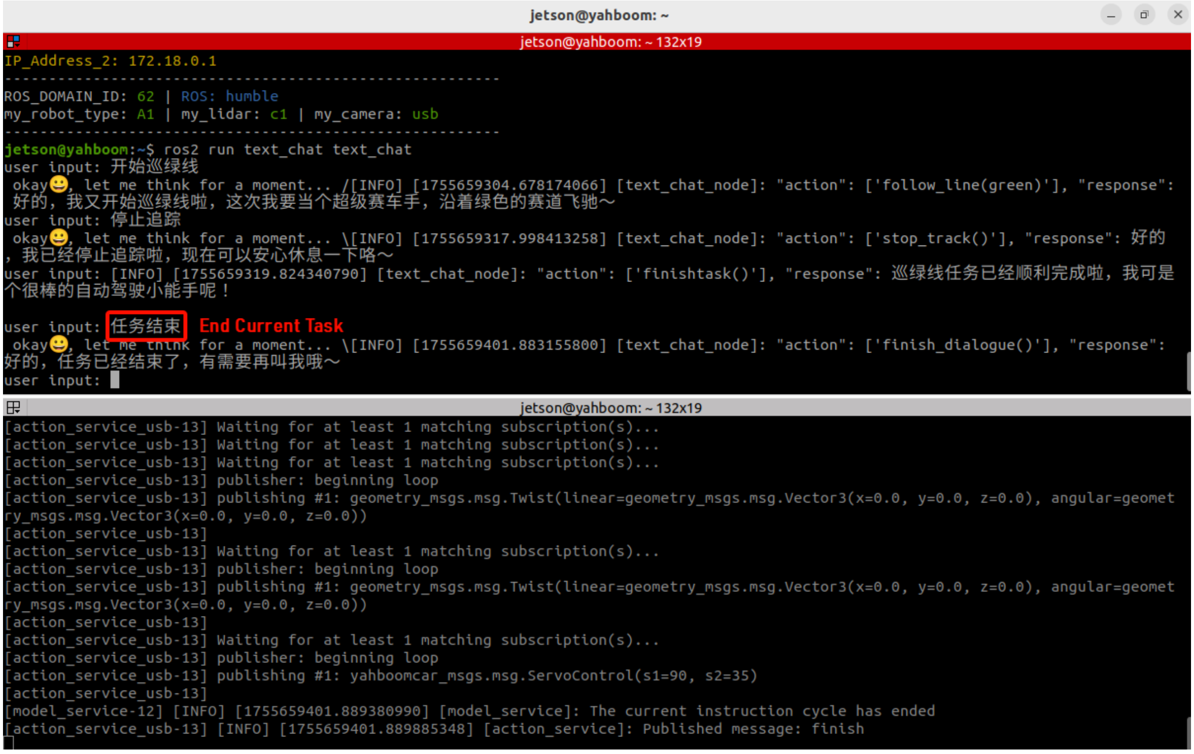
- For USB Camera users, use **[Stop Tracking]**, **[End Tracking]**, or **[Stop Line Patrol]**
- For Depth Camera users, use **[Stop Following]**, **[End Following]**, or **[Stop Line Patrol]**



```
                                              [System Information]
IP_Address_1: 192.168.11.198
IP_Address_2: 172.18.0.1
-------------------------------------------------------
ROS_DOMAIN_ID: 62 | ROS: humble
my_robot_type: A1 | my_lidar: c1 | my_camera: usb
-------------------------------------------------------
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: 开始巡绿线
 okay😀, let me think for a moment... /[INFO] [1755659304.678174066] [text_chat_node]: "action": ['follow_line(green)'], "response":
 好的，我又开始巡绿线啦，这次我要当个超级赛车手，沿着绿色的赛道飞驰～
user input: 停止追踪
 okay😀, let me think for a moment... \[INFO] [1755659317.998413258] [text_chat_node]: "action": ['stop_track()'], "response": 好的
，我已经停止追踪啦，现在可以安心休息一下咯～
user input: [INFO] [1755659319.824340790] [text_chat_node]: "action": ['finishtask()'], "response": 巡绿线任务已经顺利完成啦，我可是
个很棒的自动驾驶小能手呢！
```

```
ry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
[action_service_usb-13]
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: yahboomcar_msgs.msg.ServoControl(s1=90, s2=35)
[action_service_usb-13]
[action_service_usb-13] [INFO] [1755659306.158563458] [follow_line]: Loaded 4 color profiles
[action_service_usb-13] [INFO] [1755659306.166452513] [follow_line]: Green!
[action_service_usb-13] [INFO] [1755659318.012355475] [action_service]: Published message: 机器人反馈:执行追踪任务完成
[action_service_usb-13] [INFO] [1755659318.055850557] [action_service]: killed process_pid
[action_service_usb-13] Exception in thread Thread-354 (execute):
[action_service_usb-13] Traceback (most recent call last):
[action_service_usb-13]   File "/usr/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
[action_service_usb-13]     self.run()
[action_service_usb-13]   File "/usr/lib/python3.10/threading.py", line 953, in run
[action_service_usb-13]     self._target(*self._args, **self._kwargs)
[action_service_usb-13]   File "/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_voice_ctrl/lib/python3.10/site-pack
ages/yahboomcar_voice_ctrl/follow_line.py", line 180, in execute
[action_service_usb-13]     self.pub_cmdVel.publish(twist)
[action_service_usb-13]   File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/publisher.py", line 70, in publish
```

After completing a task, the robot enters a waiting state. In this state, commands are directly passed to the execution layer model, and all conversation history is retained. We can enter the "**End Current Task**" command again to end the robot's current task cycle and start a new one.



```
IP_Address_2: 172.18.0.1
--------------------------------------------------------
ROS_DOMAIN_ID: 62 | ROS: humble
my_robot_type: A1 | my_lidar: c1 | my_camera: usb
--------------------------------------------------------
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: 开始巡绿线
 okay😀, let me think for a moment... /[INFO] [1755659304.678174066] [text_chat_node]: "action": ['follow_line(green)'], "response":
 好的，我又开始巡绿线啦，这次我要当个超级赛车手，沿着绿色的赛道飞驰～
user input: 停止追踪
 okay😀, let me think for a moment... \[INFO] [1755659317.998413258] [text_chat_node]: "action": ['stop_track()'], "response": 好的
，我已经停止追踪啦，现在可以安心休息一下咯～
user input: [INFO] [1755659319.824340790] [text_chat_node]: "action": ['finishtask()'], "response": 巡绿线任务已经顺利完成啦，我可是
个很棒的自动驾驶小能手呢！

user input: 任务结束
 okay😀, let me think for a moment... \[INFO] [1755659401.883155800] [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: 
```

```
[action_service_usb-13] Waiting for at least 1 matching subscription(s)...
[action_service_usb-13] Waiting for at least 1 matching subscription(s)...
[action_service_usb-13] Waiting for at least 1 matching subscription(s)...
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0), angular=geomet
ry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
[action_service_usb-13]
[action_service_usb-13] Waiting for at least 1 matching subscription(s)...
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0), angular=geomet
ry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
[action_service_usb-13]
[action_service_usb-13] Waiting for at least 1 matching subscription(s)...
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: yahboomcar_msgs.msg.ServoControl(s1=90, s2=35)
[action_service_usb-13]
[model_service-12] [INFO] [1755659401.889380990] [model_service]: The current instruction cycle has ended
[action_service_usb-13] [INFO] [1755659401.889885348] [action_service]: Published message: finish
```

4. Source Code Analysis

Source code location:

Jetson Orin Nano host:

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

Jetson Nano, Raspberry Pi host:

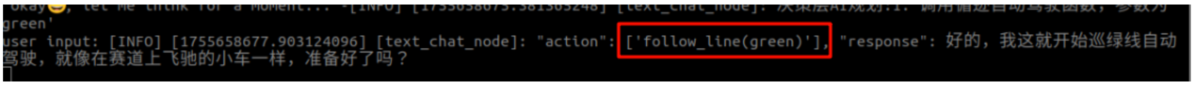
You need to first enter Docker.

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

4.1 USB Camera, action_service_usb.py

- Example 1 uses the **seewhat**, **follow_line(self, color)**, and **stop_track** methods in the **CustomActionServer** class.
  - The **seewhat** function primarily retrieves the camera's color image.
  - The **follow_line(self, color)** function performs color line tracking.
  - **stop_track()** issues a stop tracking command function.

This section focuses on the **follow_line(self, color)** function, which requires a color parameter, which can be 'red', 'green', 'blue', or 'yellow'.



```
okay😊, let me think for a moment... -[INFO] [1755658673.381363248] [text_chat_node]: 决策层AI规划:1. 调用循迹自动驾驶函数，参数为
green'
user input: [INFO] [1755658677.903124096] [text_chat_node]: "action": ['follow_line(green)'], "response": 好的，我这就开始巡绿线自动
驾驶，就像在赛道上飞驰的小车一样，准备好了吗？
```

```
# Start the line-following autonomous driving subprocess program
process_1 = subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl',
'follow_line', '--ros-args', '-p', f'target_color:={target_color}'])
```

The startup program source code path is:

`~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_voice_ctrl/yahboomcar_voice_ctrl/follow_line.py`

```
def follow_line(self,color):
    try:
        self.follow_line_future = Future()
        color = color.strip("'\"")
        if color == 'red':
            target_color = int(1)
        elif color == 'green':
            target_color = int(2)
        elif color == 'blue':
            target_color = int(3)
        elif color == 'yellow':
            target_color = int(4)
        else:
            target_color = int(1)
        process_1 = subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl',
'follow_line','--ros-args','-p',f'target_color:={target_color}'])
        while not self.follow_line_future.done():
            if self.interrupt_flag:
                break
            time.sleep(0.1)
        self.get_logger().info(f'killed process_pid')
        self.kill_process_tree(process_1.pid)
        self.cancel()
    except:
        self.get_logger().error('follow_line Startup failure')
        return
```

When the main model receives the user input of the [Stop Tracking] or [End Tracking] command,

or if the tracking target is lost for more than 10 seconds,

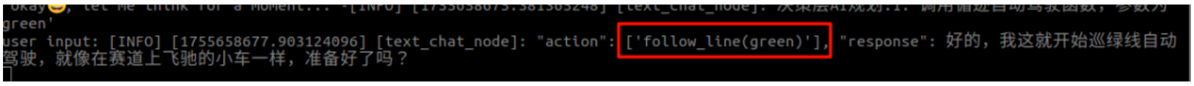
the **stop_track** method is called, sending the future.done signal. The `while not self.follow_line_future.done()` block in the **colorTrack** function then exits. The **kill_process_tree** method is then called to recursively kill the child process tree. Finally, the status of the action execution is reported to the main model at the execution layer.

4.2 nuwa Depth Camera, action_service_nuwa.py

- Example 1 uses the **seewhat**, **follow_line(self, color)**, and **stop_follow** methods in the **CustomActionServer** class.
- The **seewhat** function primarily retrieves the color image from the depth camera.
- The **follow_line(self, color)** function performs color line tracking.
- **stop_follow()** issues a stop command to follow the line.

This section focuses on the **follow_line(self, color)** function, which requires a color parameter, which can be 'red', 'green', 'blue', or 'yellow'.

```
#Start the line patrol automatic driving subprocess program
process_1= subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl_depth',
'voice_follow_line','--ros-args','-p',f'colcor:={target_color}'])
```



```
okay😊, let me think for a moment... -[INFO] [1755658675.381365248] [text_chat_node]: 决策层AI规划:1. 调用循迹自动驾驶函数，参数为
green'
user input: [INFO] [1755658677.903124096] [text_chat_node]: "action": ['follow_line(green)'], "response": 好的，我这就开始巡绿线自动
驾驶，就像在赛道上飞驰的小车一样，准备好了吗？
```

The startup program source code path is:

`~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_voice_ctrl_depth/yahboomcar_voice_ctrl_depth/follow/voice_follow_line.py`

```
def follow_line(self,color):
    self.follow_line_future = Future() #Reset Future object
    color = color.strip("'\"")  # Remove single and double quotes
    if color == 'red':
        target_color = float(1)
    elif color == 'green':
        target_color = float(2)
    elif color == 'blue':
        target_color = float(3)
    elif color == 'yellow':
        target_color = float(4)
    else:
        self.get_logger().info('color_sort:error')
        return
    process_1= subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl_depth',
'voice_follow_line','--ros-args','-p',f'colcor:={target_color}'])
    while not self.follow_line_future.done():
        if self.interrupt_flag:
            break
        time.sleep(0.1)

    self.kill_process_tree(process_1.pid)
    self.cancel()
```

When the main model receives the user input of the "Stop Following" or "End Following" command,

or if the tracking target is lost for more than 10 seconds,

the **stop_follow** method is called, sending the future.done signal. The `while not self.follow_line_future.done()` block in the **follow_line** function is then exited. The **kill_process_tree** method is then called to recursively kill the child process tree. Finally, the status of the execution action is reported to the main model at the execution layer.