

# Multimodal visual understand + PTZ tracking(Text Version)

---

## 1. Course Content

---

1. Learn to use robot visual understanding combined with gimbal tracking.
2. Analyze newly discovered key source code.

## 2. Preparation

---

### 2.1 Content Description

This course uses the Jetson Orin NANO as an example. For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the 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 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-v1-72b-instruct:free", "qwen-v1-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. If you need to increase or decrease the diversity of the large model's responses, refer to the section on configuring the decision-layer large model parameters in the **[03.AI Model Basics] -- [5.Configure AI large model]**.

⚡ It's recommended to try the previous visual example first. This example adds voice functionality to the singleton example. The functionality is largely the same, so I won't go into detail about the implementation, code debugging, and results!

## 3. Running the Example

---

### 3.1 Starting the Program

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

Open a terminal on the car and enter the command:

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

```
jetson@yahboom: ~  
-----  
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 it.

```
ros2 run text_chat text_chat
```

```
jetson@yahboom: ~  
[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 xx tracking

Color/Face/Object/Machine Code/QR Code/Gesture Recognition/Human Posture

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

Object tracking

⚠ Please do not end the text with a period or any other characters!

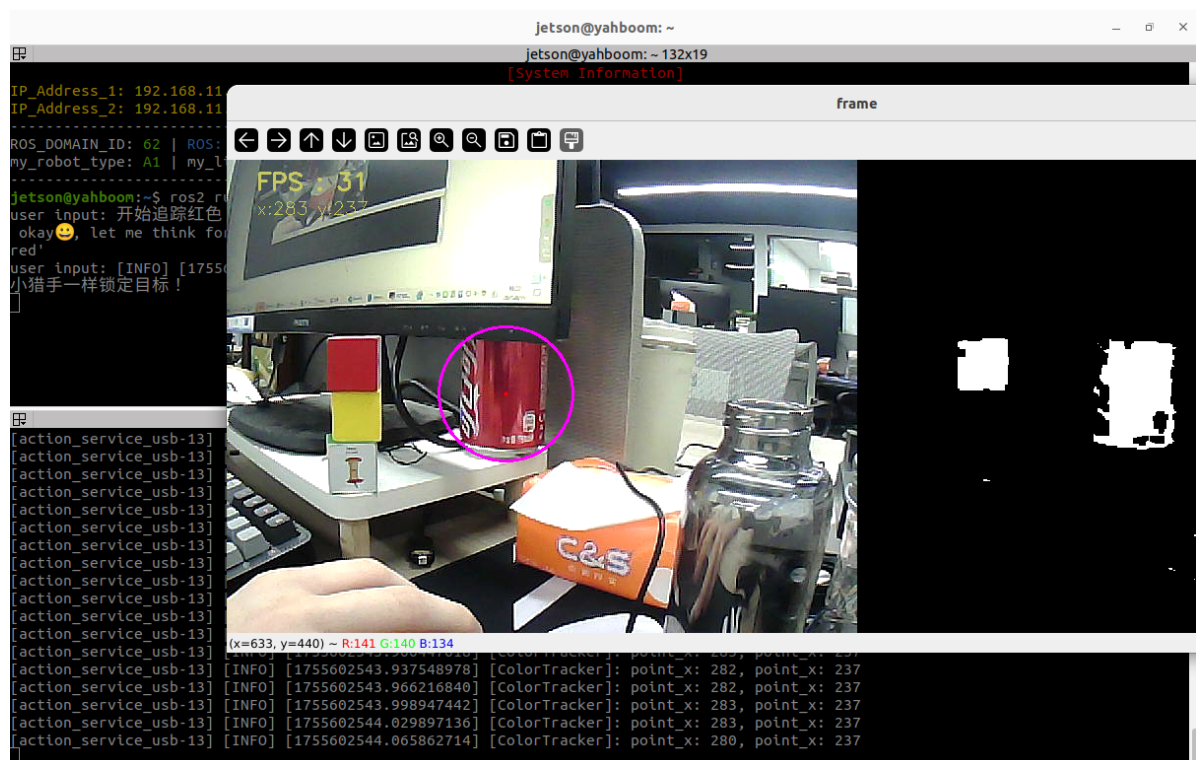
### 3.2.1 Example 1: "Start Tracking Red"

Type "Start Tracking Red" in the terminal. The terminal will print the following information:

```
jetson@yahboom: ~
jetson@yahboom: ~ 132x19
[System Information]
IP_Address_1: 192.168.11.198
IP_Address_2: 192.168.11.195
-----
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: 开始追踪红色 Start Tracking Red
okay😊, let me think for a moment... [INFO] [1755602476.070940715] [text_chat_node]: 决策层AI规划:1. 调用追踪指定颜色函数, 参数为 'red'
user input: [INFO] [1755602478.650083045] [text_chat_node]: "action": ['colorTrack(red)'], "response": 好的, 我开始追踪红色啦, 就像小猎手一样锁定目标!

[action_service_usb-13] [INFO] [1755602494.434917477] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.466952300] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.498273031] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.529225707] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.563277615] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.598497116] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.629715312] [ColorTracker]: point_x: 48, point_x: 223
[action_service_usb-13] [INFO] [1755602494.664062503] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.698996002] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.724937166] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.761194876] [ColorTracker]: point_x: 46, point_x: 223
[action_service_usb-13] [INFO] [1755602494.794039157] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.827506421] [ColorTracker]: point_x: 46, point_x: 223
[action_service_usb-13] [INFO] [1755602494.857494653] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.890564997] [ColorTracker]: point_x: 46, point_x: 223
[action_service_usb-13] [INFO] [1755602494.923390717] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.960621799] [ColorTracker]: point_x: 47, point_x: 223
[action_service_usb-13] [INFO] [1755602494.989175445] [ColorTracker]: point_x: 47, point_x: 223
```

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



Move the object slowly, and the servo and gimbal will follow.

If there is no target to track in the image, the program will count down for 10 seconds, and the terminal will print a 5-second countdown. The process will automatically end, and the task will be considered complete.

```
jetson@yahboom: ~
jetson@yahboom: ~ 132x19
[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: 开始追踪红色
okay😊, let me think for a moment... \[INFO] [1755603417.136043726] [text_chat_node]: 决策层AI规划:1. 调用追踪指定颜色函数, 参数为 'red'
user input: \[INFO] [1755603420.431562347] [text_chat_node]: "action": ['colorTrack(red)'], "response": 好呀, 我现在开始追踪红色物体啦, 就像一只专注的小猎手~
\[INFO] [1755603640.336756742] [text_chat_node]: "action": ['finishtask()'], "response": 我已经成功开始追踪红色物体啦, 任务完成!
\[\n
jetson@yahboom: ~ 132x19
[action_service_usb-13] \[INFO] [1755603634.304228518] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.335971724] [ColorTracker]: 5
[action_service_usb-13] \[INFO] [1755603634.338274729] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.371462921] [ColorTracker]: 5
[action_service_usb-13] \[INFO] [1755603634.372168358] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.407411769] [ColorTracker]: 5
[action_service_usb-13] \[INFO] [1755603634.408573256] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.436909220] [ColorTracker]: 4
[action_service_usb-13] \[INFO] [1755603634.439065147] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.471978448] [ColorTracker]: 4
[action_service_usb-13] \[INFO] [1755603634.472682381] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.504477300] [ColorTracker]: 4
[action_service_usb-13] \[INFO] [1755603634.506719951] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.540203291] [ColorTracker]: 4
[action_service_usb-13] \[INFO] [1755603634.540932537] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.567198944] [ColorTracker]: 4
[action_service_usb-13] \[INFO] [1755603634.568153703] [ColorTracker]: point_x: 592, point_x: 63
[action_service_usb-13] \[INFO] [1755603634.600191896] [ColorTracker]: 4
[action_service_usb-13] \[INFO] [1755603634.600942007] [ColorTracker]: point_x: 592, point_x: 63
```

To manually end the task, press **ENTER** in the terminal. Press the key to continue the dialog input, **[Stop Tracing]** or **[End Tracing]**

```
Jetson@yahboom: ~
jetson@yahboom: ~ 132x19
IP_Address_1: 192.168.11.198
IP_Address_2: 192.168.11.195
-----
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: 开始追踪红色
okay😊, let me think for a moment... \[INFO] [1755602476.070940715] [text_chat_node]: 决策层AI规划:1. 调用追踪指定颜色函数, 参数为 'red'
user input: \[INFO] [1755602478.650083045] [text_chat_node]: "action": ['colorTrack(red)'], "response": 好的, 我开始追踪红色啦, 就像小猎手一样锁定目标!
user input: 停止追踪 Stop Tracing
okay😊, let me think for a moment... \[INFO] [1755602577.708736866] [text_chat_node]: "action": ['stop_track()'], "response": 收到指令, 我已停止追踪, 现在可以专心做别的事情啦!
user input: \[INFO] [1755602579.409938039] [text_chat_node]: "action": ['finishtask()'], "response": 追踪任务已经顺利完成, 我随时准备接受新指令哦!
\[\n
jetson@yahboom: ~ 132x19
[action_service_usb-13] \[INFO] [1755602577.460714001] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.492397455] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.531752718] [ColorTracker]: point_x: 286, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.559713746] [ColorTracker]: point_x: 288, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.589833109] [ColorTracker]: point_x: 286, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.624135468] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.655091423] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.692506733] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.716749239] [action_service]: Published message: 机器人反馈:执行追踪任务完成
[action_service_usb-13] \[INFO] [1755602577.722256730] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.753514719] [ColorTracker]: point_x: 287, point_x: 230
[action_service_usb-13] \[INFO] [1755602577.762157048] [action_service]: killed process_pid
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: yahboomcar_msgs.msg.ServoControl(s1=90, s2=30)
[action_service_usb-13] \n
[action_service_usb-13] import done
[action_service_usb-13] init done
[action_service_usb-13] start it
[action_service_usb-13] shutting down this node.
```

🔗 If the following warning appears when ending tracing, this is normal. It indicates that the child process has been killed and will not affect normal operation of the program.

```
[action_service_usb-13] Failed to publish log message to rosout: publisher's context is invalid, at ./src/rc1/publisher.c:389
[action_service_usb-13] Exception in thread Thread-284 (execute):
[action_service_usb-13] rc1py._rc1py_pybind11.RCLError: Failed to publish: publisher's context is invalid, at ./src/rc1/publisher.c:389
```

After completing a task, the robot enters a waiting state. Instructions are passed directly to the execution layer model, and all conversation history is retained. You can enter the "**End current task**" command again to end the current task cycle and start a new one.

```
user input: 开始追踪绿色
okay😊 let me think for a moment... \[INFO] [1755602723.253393354] [text_chat_node]: "action": ['colorTrack(green)'], "response":
好的, 现在我开始追踪绿色啦, 眼睛已经锁定目标, 就像小侦探一样专注!
user input: 结束追踪
okay😊 let me think for a moment... \[INFO] [1755602737.046441725] [text_chat_node]: "action": ['stop_track()'], "response": 收到
, 我已经停止追踪啦, 现在可以放松一下眼睛了~
user input: \[INFO] [1755602738.141789123] [text_chat_node]: "action": ['finishtask()'], "response": 追踪任务已经顺利完成, 我随时准备
接受新指令哦!
user input: 任务结束 End current task
okay😊 let me think for a moment... \[INFO] [1755603185.200661916] [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:

jetson@yahboom: ~ 132x19
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: yahboomcar_msgs.msg.ServoControl(s1=90, s2=30)
[action_service_usb-13]
[action_service_usb-13] import done
[action_service_usb-13] init done
[action_service_usb-13] start it
[action_service_usb-13] Shutting down this node.
[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] 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=30)
[model_service-12] \[INFO] [1755603185.208134502] [model_service]: The current instruction cycle has ended
[action_service_usb-13] \[INFO] [1755603185.208300016] [action_service]: Published message: finish
```

### 3.2.2 Example 2: "Please track the object in my hand"

⚠ The coordinates obtained in this example are derived entirely from the inference of a large AI model. Therefore, it is recommended to use a newer model for better results!

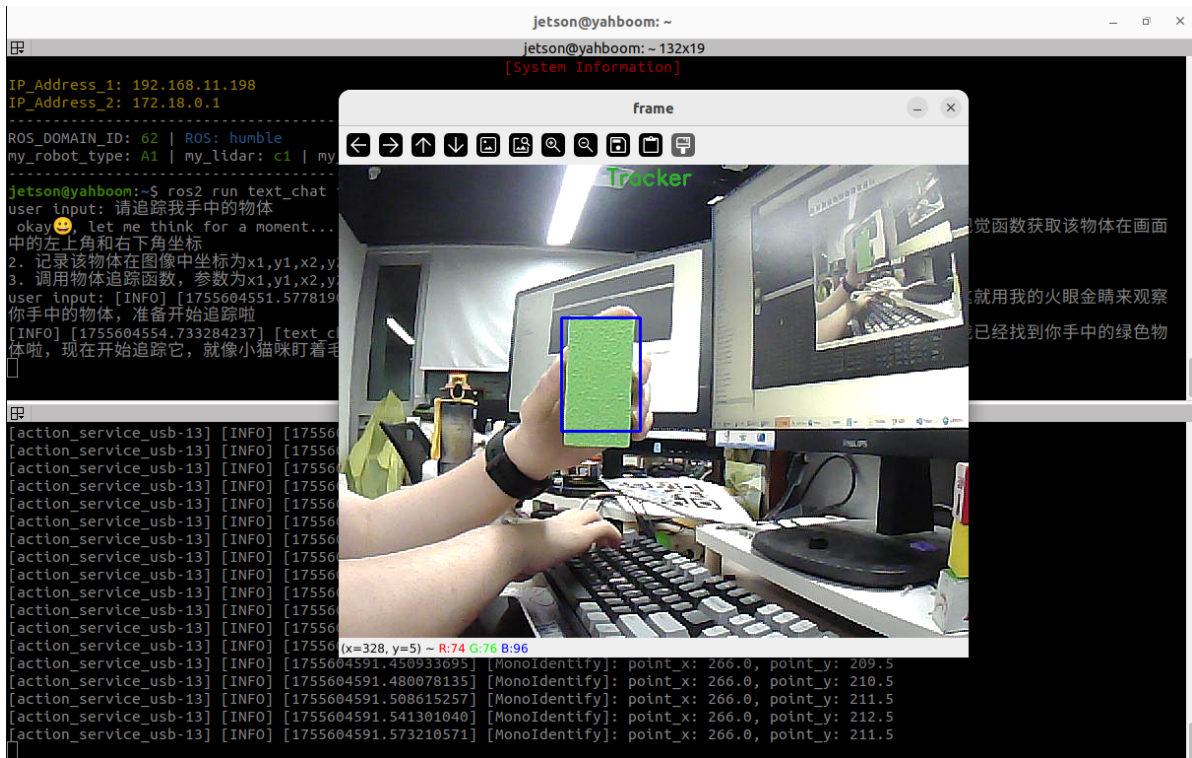
Type "Please track the object in my hand" in the terminal. The terminal prints the following information:

```
jetson@yahboom: ~
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: 请追踪我手中的物体 Please track the object in my hand
okay😊 let me think for a moment... \[INFO] [1755604548.570634479] [text_chat_node]: 决策层AI规划: 1. 调用视觉函数获取该物体在画面
中的左上角和右下角坐标
2. 记录该物体在图像中坐标为x1,y1,x2,y2
3. 调用物体追踪函数, 参数为x1,y1,x2,y2
user input: \[INFO] [1755604551.577819086] [text_chat_node]: "action": ['seewatch()'], "response": 好的呀, 我这就用我的火眼金睛来观察
你手中的物体, 准备开始追踪啦
\[INFO] [1755604554.733284237] [text_chat_node]: "action": ['monoTracker(240, 145, 320, 260)'], "response": 我已经找到你手中的绿色物
体啦, 现在开始追踪它, 就像小猫眯盯着毛线球一样专注

jetson@yahboom: ~ 132x19
[action_service_usb-13] \[INFO] [1755604563.847463367] [MonoIdentify]: point_x: 282.0, point_y: 205.5
[action_service_usb-13] \[INFO] [1755604563.885402776] [MonoIdentify]: point_x: 282.0, point_y: 206.5
[action_service_usb-13] \[INFO] [1755604563.908191402] [MonoIdentify]: point_x: 282.0, point_y: 205.5
[action_service_usb-13] \[INFO] [1755604563.940599508] [MonoIdentify]: point_x: 282.0, point_y: 206.5
[action_service_usb-13] \[INFO] [1755604563.967903207] [MonoIdentify]: point_x: 282.0, point_y: 205.5
[action_service_usb-13] \[INFO] [1755604564.000194606] [MonoIdentify]: point_x: 282.0, point_y: 204.5
[action_service_usb-13] \[INFO] [1755604564.031078946] [MonoIdentify]: point_x: 282.0, point_y: 203.5
[action_service_usb-13] \[INFO] [1755604564.062571627] [MonoIdentify]: point_x: 282.0, point_y: 204.5
[action_service_usb-13] \[INFO] [1755604564.088901020] [MonoIdentify]: point_x: 282.0, point_y: 203.5
[action_service_usb-13] \[INFO] [1755604564.121282822] [MonoIdentify]: point_x: 282.0, point_y: 204.5
[action_service_usb-13] \[INFO] [1755604564.148733438] [MonoIdentify]: point_x: 282.0, point_y: 203.5
[action_service_usb-13] \[INFO] [1755604564.178564396] [MonoIdentify]: point_x: 280.0, point_y: 204.5
[action_service_usb-13] \[INFO] [1755604564.211673200] [MonoIdentify]: point_x: 280.0, point_y: 203.5
[action_service_usb-13] \[INFO] [1755604564.238469393] [MonoIdentify]: point_x: 280.0, point_y: 204.5
[action_service_usb-13] \[INFO] [1755604564.271315020] [MonoIdentify]: point_x: 280.0, point_y: 203.5
[action_service_usb-13] \[INFO] [1755604564.298612607] [MonoIdentify]: point_x: 280.0, point_y: 204.5
[action_service_usb-13] \[INFO] [1755604564.329501971] [MonoIdentify]: point_x: 280.0, point_y: 203.5
[action_service_usb-13] \[INFO] [1755604564.361098081] [MonoIdentify]: point_x: 280.0, point_y: 204.5
```

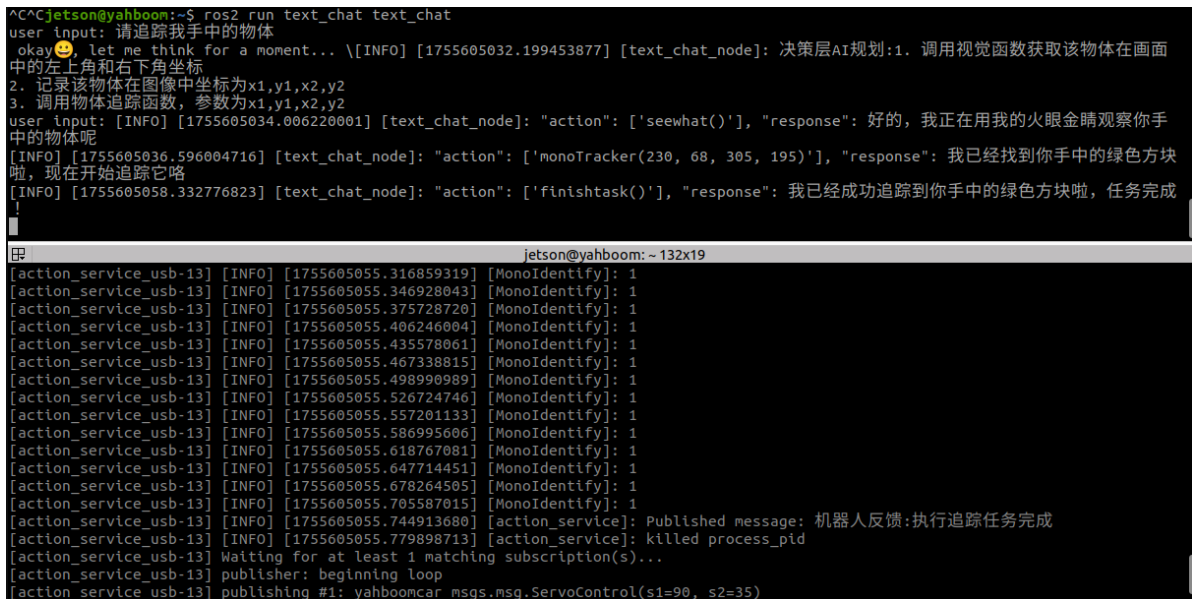
Please hold any object in your hand and place it in the field of view until the tracking frame appears. A window titled **frame** will open on the VNC screen to display the image from the current robot's perspective.





Slowly move the object, and the servo gimbal will follow.

If there's no target to track in the image, the program will count down for 10 seconds, and the terminal will print a 5-second countdown. The process will automatically end, and the mission will be considered complete.



To manually end the task, press **ENTER** in the terminal, continue the dialog input, **[Stop Tracking]** or **[End Tracking]**

```
jetson@yahboom: ~
jetson@yahboom: ~ 132x19
my_robot_type: A1 | my_lidar: c1 | my_camera: usb
-----
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: 请追踪我手中的物体
okay 😊 let me think for a moment... -[INFO] [1755604548.570634479] [text_chat_node]: 决策层AI规划:1. 调用视觉函数获取该物体在画面中的左上角和右下角坐标
2. 记录该物体在图像中坐标为x1,y1,x2,y2
3. 调用物体追踪函数, 参数为x1,y1,x2,y2
user input: [INFO] [1755604551.577819086] [text_chat_node]: "action": ['seewhat()'], "response": 好的呀, 我这就用我的火眼金睛来观察你手中的物体, 准备开始追踪啦
[INFO] [1755604554.733284237] [text_chat_node]: "action": ['monoTracker(240, 145, 320, 260)'], "response": 我已经找到你手中的绿色物体啦, 现在开始追踪它, 就像小猫眯盯着毛线球一样专注
user input: 结束追踪 Stop Tracing
okay 😊 let me think for a moment... \[INFO] [1755604869.456603972] [text_chat_node]: "action": ['stop_track()'], "response": 好的, 我已经停止追踪啦, 随时可以再叫我哦
user input: [INFO] [1755604871.641374829] [text_chat_node]: "action": ['finish_dialogue()'], "response": 追踪任务已经结束啦, 有需要再叫我哦

jetson@yahboom: ~ 132x19
[action_service_usb-13] [INFO] [1755604869.492016500] [MonoIdentify]: point x: 312.0, point y: 222.5
[action_service_usb-13] [INFO] [1755604869.497511933] [action_service]: killed process_pid
[action_service_usb-13] [INFO] [1755604869.520814993] [MonoIdentify]: point x: 312.0, point y: 221.5
[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] OpenCV Version: 4.11.0
[action_service_usb-13] Shutting down this node.
[model_service-12] [INFO] [1755604871.646513513] [model_service]: The current instruction cycle has ended
[action_service_usb-13] [INFO] [1755604871.646853909] [action_service]: Published message: finish
[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=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
[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]
```

After completing a task, the robot enters a waiting state. Instructions are passed directly to the execution layer model, and all conversation history is retained. You can enter the "End current task" command again to end the current task cycle and start a new one.

```
user input: 开始追踪绿色
okay 😊 let me think for a moment... [INFO] [1755602723.253393354] [text_chat_node]: "action": ['colorTrack(green)'], "response": 好的, 现在我开始追踪绿色啦, 眼睛已经锁定目标, 就像小侦探一样专注!
user input: 结束追踪
okay 😊 let me think for a moment... [INFO] [1755602737.046441725] [text_chat_node]: "action": ['stop_track()'], "response": 收到, 我已经停止追踪啦, 现在可以放松一下眼睛了~
user input: [INFO] [1755602738.141789123] [text_chat_node]: "action": ['finishtask()'], "response": 追踪任务已经顺利完成, 我随时准备接受新指令哦!

user input: 任务结束 End current task
okay 😊 let me think for a moment... \[INFO] [1755603185.200661916] [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:

jetson@yahboom: ~ 132x19
[action_service_usb-13] publisher: beginning loop
[action_service_usb-13] publishing #1: yahboomcar_msgs.msg.ServoControl(s1=90, s2=30)
[action_service_usb-13] import done
[action_service_usb-13] init done
[action_service_usb-13] start it
[action_service_usb-13] Shutting down this node.
[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=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
[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=30)
[action_service_usb-13]
[model_service-12] [INFO] [1755603185.208134502] [model_service]: The current instruction cycle has ended
[action_service_usb-13] [INFO] [1755603185.208350016] [action_service]: Published message: finish
```

## 4.Source code analysis

Source code located at:

Jetson Nano host:

```
/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.

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/largemodel/action_service_
usb.py
```

## 4.1 Example 1

action\_service\_usb.py:

In Example 1, the **seewhat**, **color\_follow**, and **stop\_track** methods in the **CustomActionServer** class are used.

- The **seewhat** function primarily retrieves the camera's color image.
- The **colorTrack(self, color)** function performs color tracking.
- The **stop\_track()** function issues a stop tracking command.

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

```
ser input: 开始追踪红色
okay 😊, let me think for a moment... \[INFO] [1755602476.070940715] [text_chat_node]: 决策层AI规划:1. 调用追踪指定颜色函数, 参数为'
ed'
ser input: [INFO] [1755602478.650083045] [text_chat_node]: "action": ["colorTrack(red)"], "response": "好的, 我开始追踪红色啦, 就像
猎手一样锁定目标!"
```

```
#Start the color tracking subprocess program
process_1 = subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl',
'colorTracker', '--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/
colorTracker.py
```

```
def colorTrack(self,color):
    try:
        self.colorTracker_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',
'colorTracker', '--ros-args', '-p', f'target_color:={target_color}'])
        while not self.colorTracker_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('colorTrack Startup failure')
    return
```



When the large model receives the user input of the [Stop Tracking] or [End Tracking] command, or when the tracking target is lost for more than 10 seconds,

The **stop\_track** method will be called to send the future.done signal. After that, `while not self.colorTracker_future.done()` in the **colorTrack** function will exit the blocking state. Then the **kill\_process\_tree** method will be called to recursively kill the process tree of the child process. Finally, the status of the execution action will be fed back to the execution layer model.

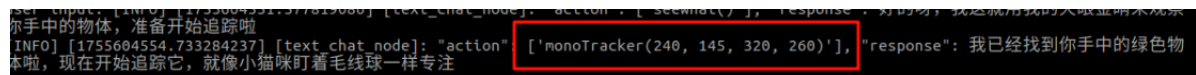
## 4.2 Example 2

action\_service\_usb.py Program:

In Example 2, the **seewhat**, **monoTracker**, and **stop\_track** methods in the **CustomActionServer** class are used.

- The **seewhat** function primarily obtains the camera's color image.
- The **monoTracker(self,x1,y1,x2,y2)** function performs object tracking.
- The **stop\_track()** function issues a stop tracking command.

The **seewhat** function primarily obtains the camera's color image. The **monoTracker(self,x1,y1,x2,y2)** function takes as parameters the coordinates of the upper-left and lower-right vertices of the object's bounding box to be tracked (the upper-left corner of the image is the pixel origin). For example, the coordinates of the outer bounding box of the green square identified in Example 2 can be found in the response from the large model: the upper-left corner is (240,145) and the lower-right corner is (320,260).



```
#Start the object tracking subprocess program
process_1 = subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl',
                              'monoTracker', '--ros-args', '-p', f'x1:={x1}', '-p', f'y1:={y1}', '-p', f'x2:={x2}', '-p', f'y2:={y2}'])
```

The startup program source code path is:

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

```
def monoTracker(self, x1, y1, x2, y2):
    try:
        self.monoTracker_future = Future()
        x1 = int(x1)
        x2 = int(x2)
        y1 = int(y1)
        y2 = int(y2)
        process_1 = subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl',
                                      'monoTracker', '--ros-args', '-p', f'x1:={x1}', '-p', f'y1:={y1}', '-p', f'x2:={x2}', '-p', f'y2:={y2}'])
        while not self.monoTracker_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('monoTracker Startup failure')  
    return
```

When the large model receives the user input of the [Stop Tracking] or [End Tracking] command, or when the tracking target is lost for more than 10 seconds,

The **stop\_track** method will be called to send the future.done signal. After that, `while not self.monoTracker_future.done()` in the **monoTracker** function will exit the blocking state. Then the **kill\_process\_tree** method will be called to recursively kill the process tree of the child process. Finally, the status of the execution action will be fed back to the execution layer model.