Multimodal visual understand + PTZ tracking(Voice Version)

1. Course Content

1. Learn to use the robot's visual understanding capabilities
2. Learn new key source code

2. Preparation

2.1 Content Description

> This course uses the Jetson Orin Nano as an example. For users of the gimbal servo USB camera version, this is used 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 **[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 big 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 big model's responses, refer to the section on configuring the decision-making big 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, and while the functionality is largely the same, I will not elaborate on the program implementation, code debugging, or results in detail! ! !

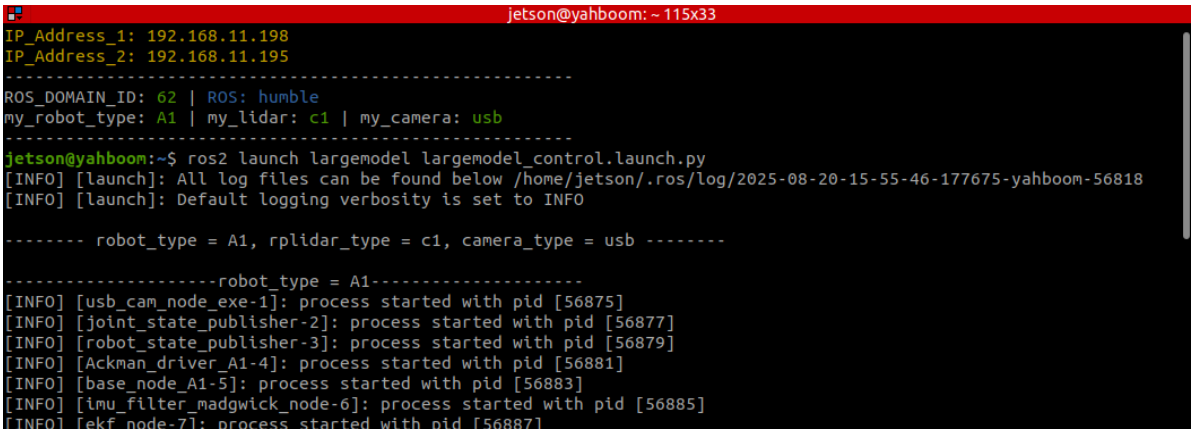
3. Running the Example

3.1 Starting the Program

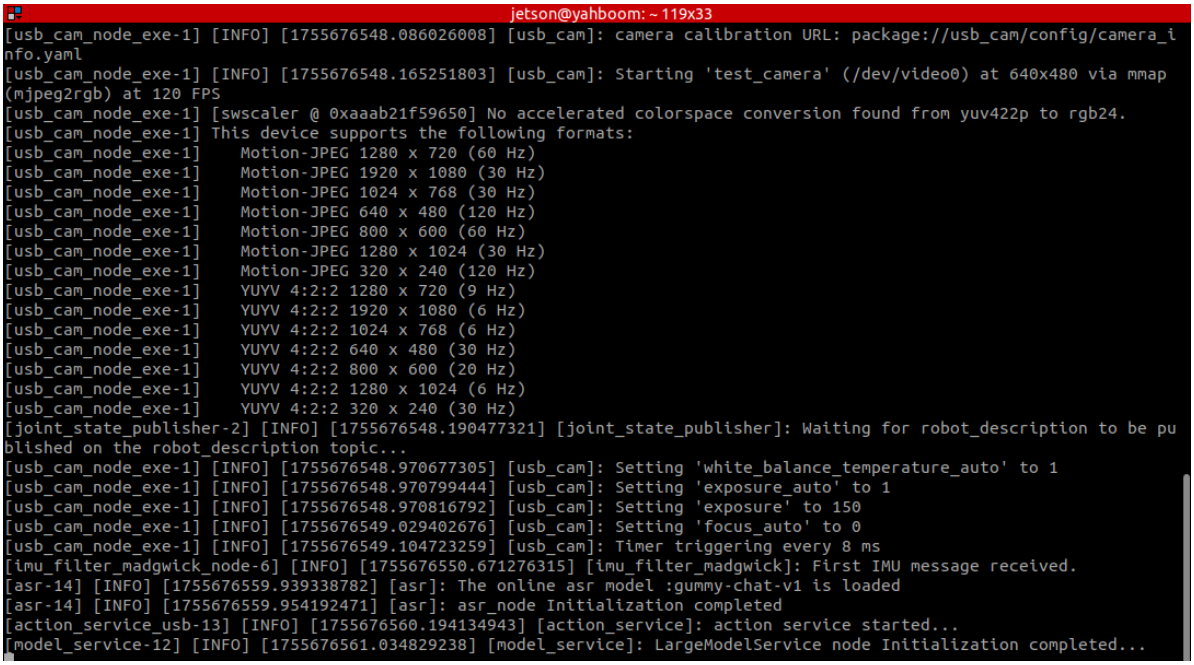
For Raspberry Pi 5 and Jetson Nano controllers, you must first enter the Docker container. For the Orin board, this is not necessary.

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

```
ros2 launch largemodel largemodel_control.launch.py
```



After initialization is complete, the following content will be displayed.



3.2 Test Case

This is a reference test case; 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

3.2.1 Example 1: "Start Tracking Red"

First, wake the robot with "Hi, yahboom" The robot responds: "I'm here, please." After the robot responds, the buzzer beeps briefly (beep—). The user can then speak. The robot will detect sound activity, printing 1 if there is sound activity and - if there is no sound activity. When the speech ends, it detects the end of the voice and stops recording if there is silence for more than 450ms.

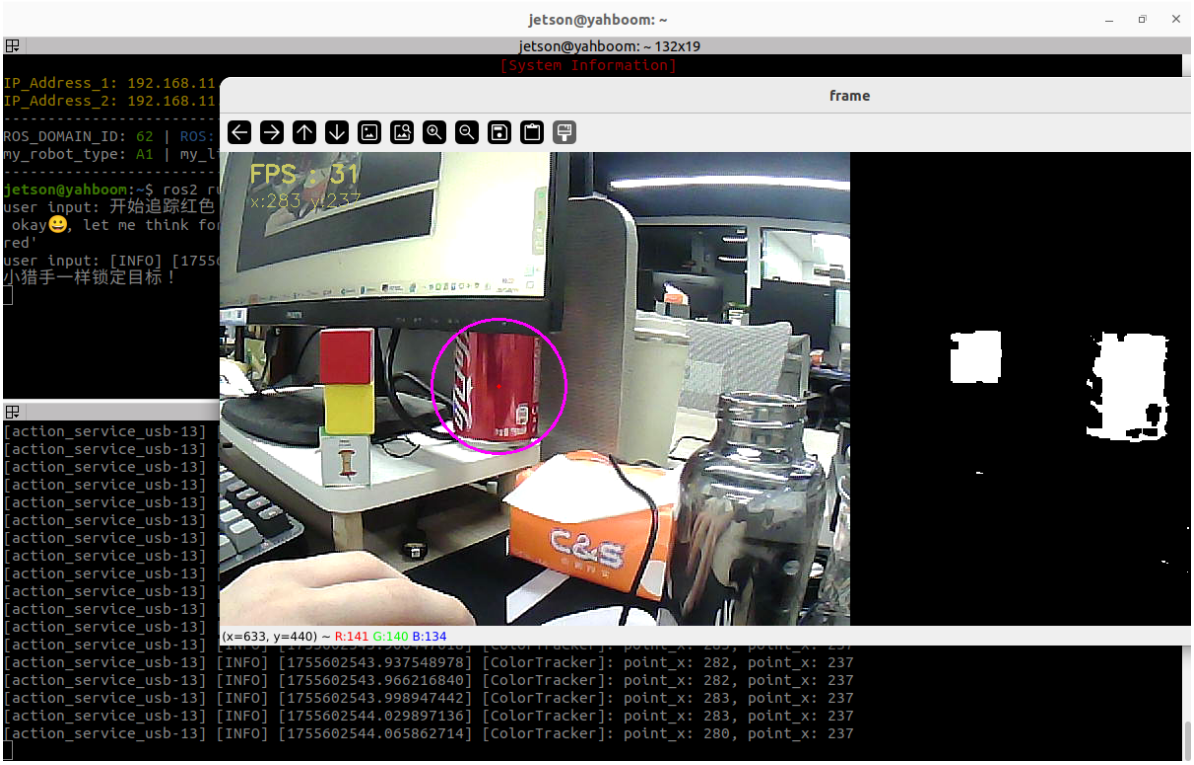
The following image shows Voice Active Detection (VAD):

```
jetson@yahboom: ~ 119x33
[action_service_usb-13] [INFO] [1755676560.194134943] [action_service]: action service started...
[model_service-12] [INFO] [1755676561.034829238] [model_service]: LargeModelService node Initialization completed...
[asr-14] [INFO] [1755677193.758468968] [asr]: I'm here
[asr-14] Cannot connect to server socket err = No such file or directory
[asr-14] Cannot connect to server request channel
[asr-14] jack server is not running or cannot be started
[asr-14] JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
[asr-14] JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
[asr-14] Cannot connect to server socket err = No such file or directory
[asr-14] Cannot connect to server request channel
[asr-14] jack server is not running or cannot be started
[asr-14] JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
[asr-14] JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
[asr-14] Cannot connect to server socket err = No such file or directory
[asr-14] Cannot connect to server request channel
[asr-14] jack server is not running or cannot be started
[asr-14] JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
[asr-14] JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
[asr-14] [INFO] [1755677195.808943354] [asr]: 1
[asr-14] [INFO] [1755677195.869161395] [asr]: 1
[asr-14] [INFO] [1755677195.930788959] [asr]: -
[asr-14] [INFO] [1755677195.990529768] [asr]: -
[asr-14] [INFO] [1755677196.051414359] [asr]: -
[asr-14] [INFO] [1755677196.110416196] [asr]: -
[asr-14] [INFO] [1755677196.169515509] [asr]: -
[asr-14] [INFO] [1755677196.230725039] [asr]: -
[asr-14] [INFO] [1755677196.291074443] [asr]: -
[asr-14] [INFO] [1755677196.351821782] [asr]: -
[asr-14] [INFO] [1755677196.421332818] [asr]: -
[asr-14] [INFO] [1755677196.482213345] [asr]: -
[asr-14] [INFO] [1755677196.542212305] [asr]: -
[asr-14] [INFO] [1755677196.603797368] [asr]: -
[asr-14] [INFO] [1755677196.664392989] [asr]: -
```

The robot will first communicate with the user, then respond to the user's instructions. The following information will be printed on the terminal:

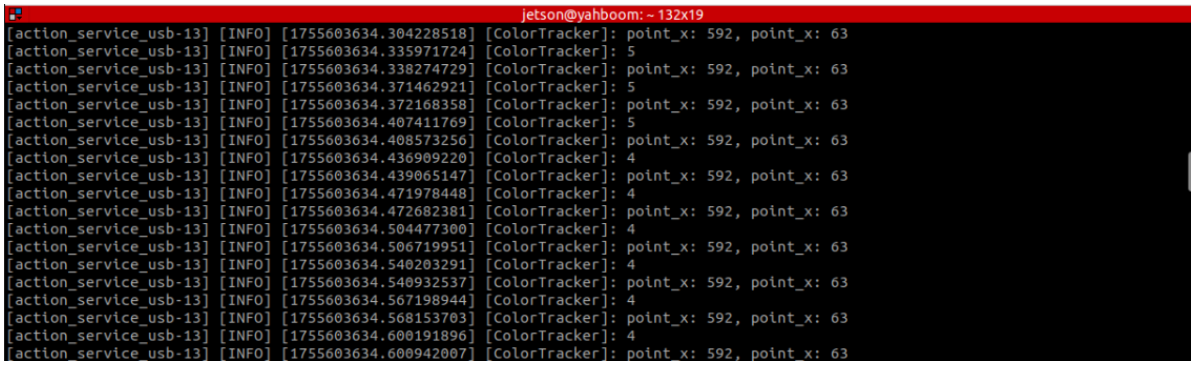
```
jetson@yahboom: ~ 122x34
[asr-14] [INFO] [1755689693.099078679] [asr]: -
[asr-14] [INFO] [1755689693.159681353] [asr]: -
[asr-14] [INFO] [1755689693.220133530] [asr]: -
[asr-14] [INFO] [1755689693.281055693] [asr]: -
[asr-14] [INFO] [1755689693.341707646] [asr]: -
[asr-14] [INFO] [1755689693.401669199] [asr]: -
[asr-14] [INFO] [1755689693.461263967] [asr]: -
[asr-14] [INFO] [1755689693.521016240] [asr]: -
[asr-14] [INFO] [1755689693.580646656] [asr]: -
[asr-14] [INFO] [1755689693.642947060] [asr]: -
[asr-14] [INFO] [1755689694.520350516] [asr]: 开始追踪红色。
[asr-14] [INFO] [1755689694.521760150] [asr]: 😀okay, let me think for a moment...
[model_service-12] [INFO] [1755689695.655800387] [model_service]: 决策层AI规划:The model service is abnormal. Check the la
rge model account or configuration options
[model_service-12] [INFO] [1755689697.777113963] [model_service]: "action": ['colorTrack(red)'], "response": 好的，我开始
追踪红色啦，就像一只专注的小猎手～
[action_service_usb-13] [INFO] [1755689704.504968251] [ColorTracker]: Loaded 4 color profiles
[action_service_usb-13] [INFO] [1755689704.509176996] [ColorTracker]: Red!
[action_service_usb-13] [INFO] [1755689704.556825478] [ColorTracker]: point_x: 15, point_x: 311
[action_service_usb-13] [INFO] [1755689704.832131391] [ColorTracker]: point_x: 15, point_x: 310
[action_service_usb-13] [INFO] [1755689704.848711617] [ColorTracker]: point_x: 17, point_x: 313
[action_service_usb-13] [INFO] [1755689704.874680759] [ColorTracker]: point_x: 17, point_x: 314
[action_service_usb-13] [INFO] [1755689704.910643073] [ColorTracker]: point_x: 15, point_x: 311
[action_service_usb-13] [INFO] [1755689704.944513991] [ColorTracker]: point_x: 17, point_x: 313
[action_service_usb-13] [INFO] [1755689704.973877124] [ColorTracker]: point_x: 16, point_x: 311
[action_service_usb-13] [INFO] [1755689705.007920650] [ColorTracker]: point_x: 16, point_x: 312
[action_service_usb-13] [INFO] [1755689705.044074583] [ColorTracker]: point_x: 16, point_x: 312
[action_service_usb-13] [INFO] [1755689705.073994103] [ColorTracker]: point_x: 16, point_x: 311
[action_service_usb-13] [INFO] [1755689705.108018656] [ColorTracker]: point_x: 16, point_x: 311
[action_service_usb-13] [INFO] [1755689705.138654017] [ColorTracker]: point_x: 16, point_x: 311
[action_service_usb-13] [INFO] [1755689705.179925945] [ColorTracker]: point_x: 16, point_x: 311
[action_service_usb-13] [INFO] [1755689705.205471887] [ColorTracker]: point_x: 16, point_x: 311
[action_service_usb-13] [INFO] [1755689705.241179067] [ColorTracker]: point_x: 16, point_x: 311
```

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

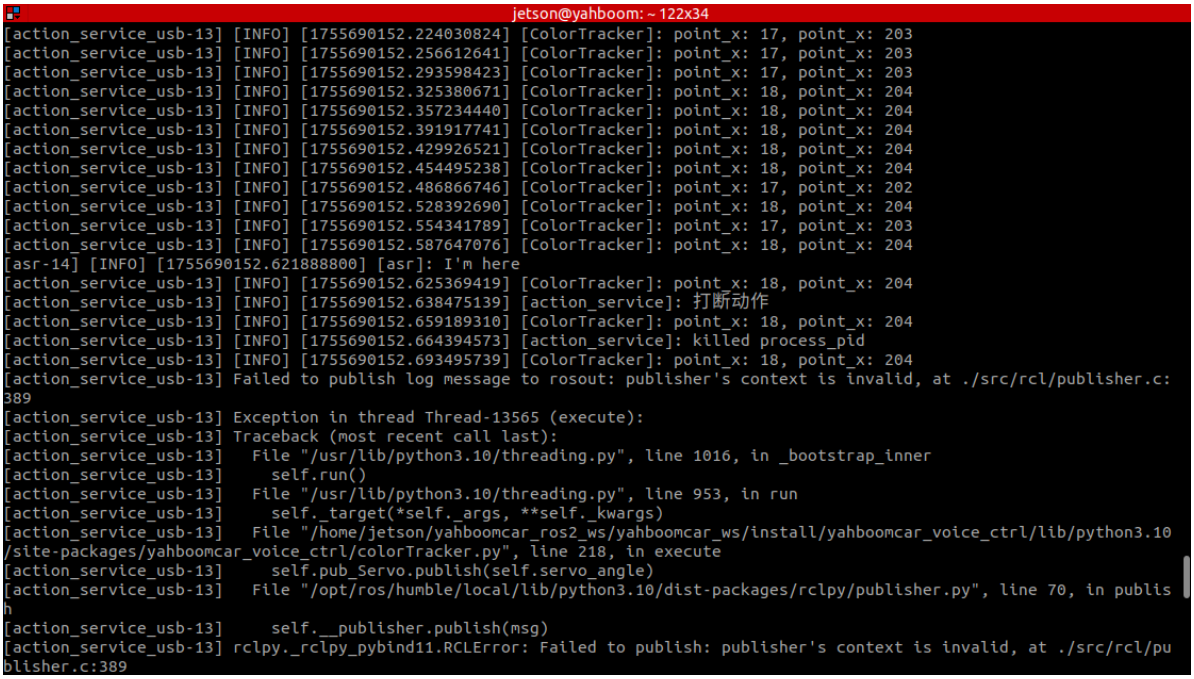


Move the object slowly, and the servo and gimbal will follow.

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



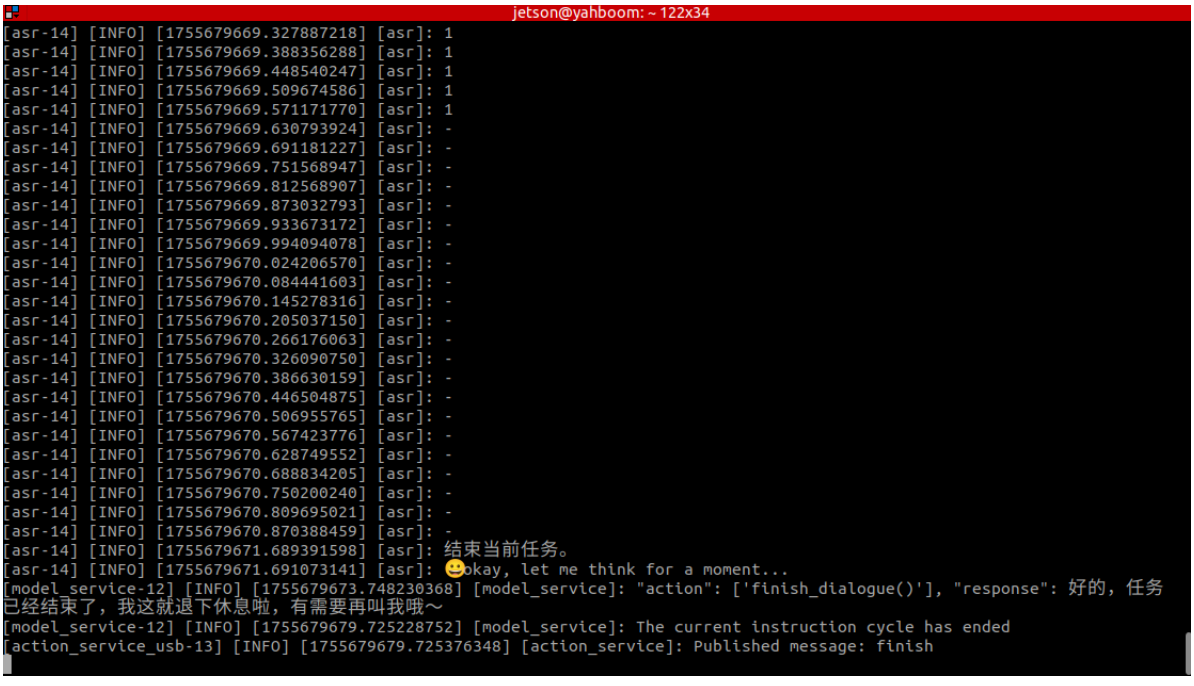
To manually end a task, wake the robot by saying "Hi, yahboom" The robot will respond, "I'm here, please." This will interrupt the program and automatically terminate the process, allowing you to proceed to the next command.



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

```
[action_service_usb-13] Failed to publish log message to rosout: publisher's
context is invalid, at ./src/rcl/publisher.c:389
[action_service_usb-13] Exception in thread Thread-284 (execute):
[action_service_usb-13] rclpy._rclpy_pybind11.RCLError: Failed to publish:
publisher's context is invalid, at ./src/rcl/publisher.c:389
```

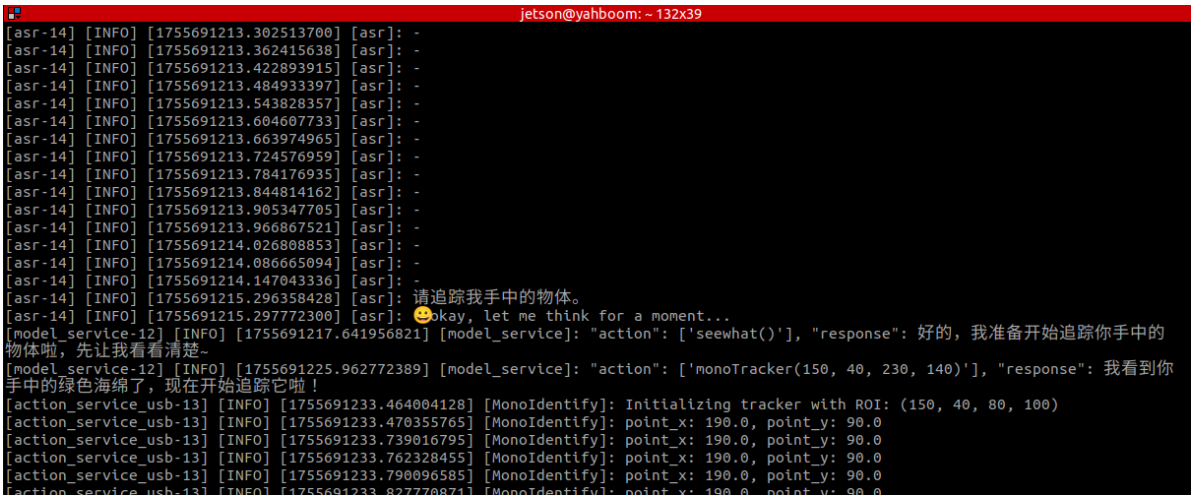
The robot now enters the free conversation state again, but all conversation history is retained. You can wake yahboom up again and use "End Current Task" to terminate the current task cycle, clear the conversation history, and start a new one.



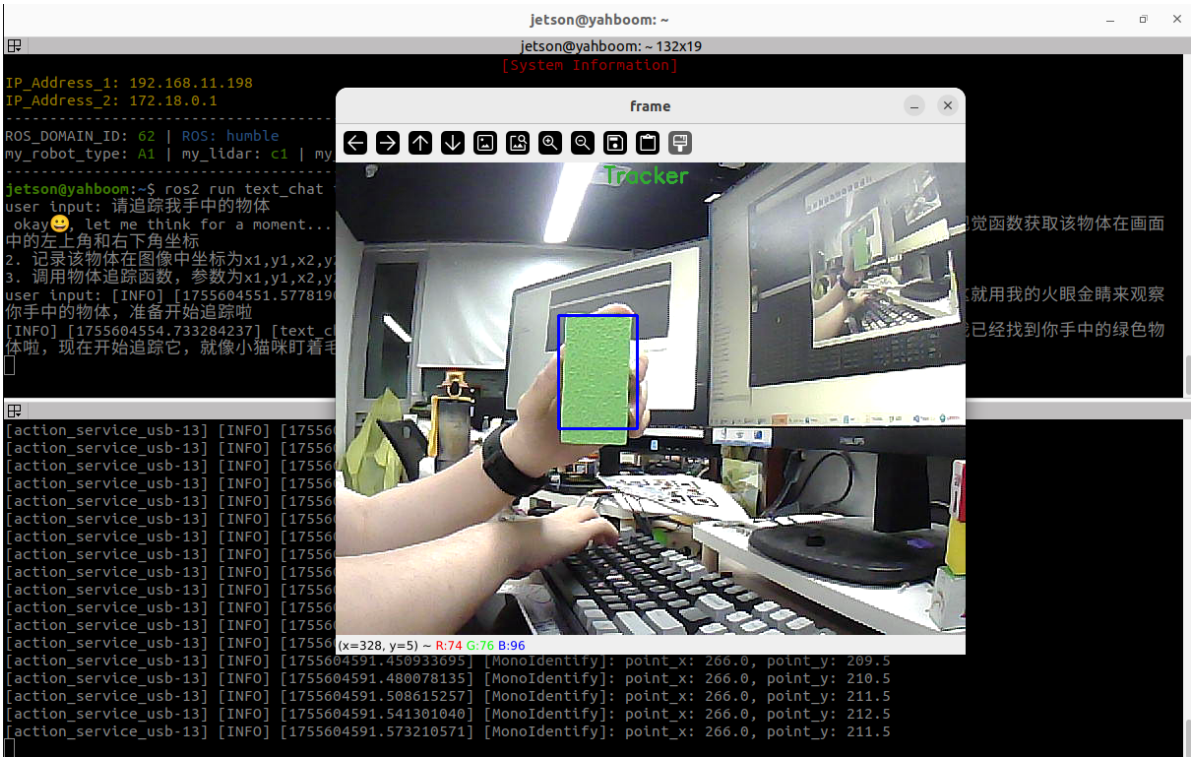
3.2.2 Example 2: "Please track the object in my hand"

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

Similar to the test in Example 1, first wake the robot with "Hi, yahboom" After the robot responds, the buzzer will briefly beep. The user can then speak. After speaking, the robot will respond and move according to the user's instructions. Hold any object in its field of view and hold until the tracking box appears.

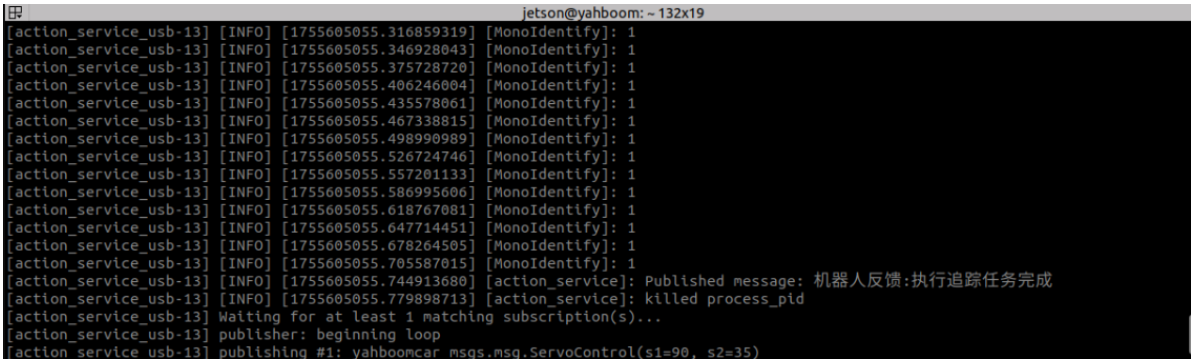


A window titled **frame** will open on the VNC screen, displaying the current robot's view.

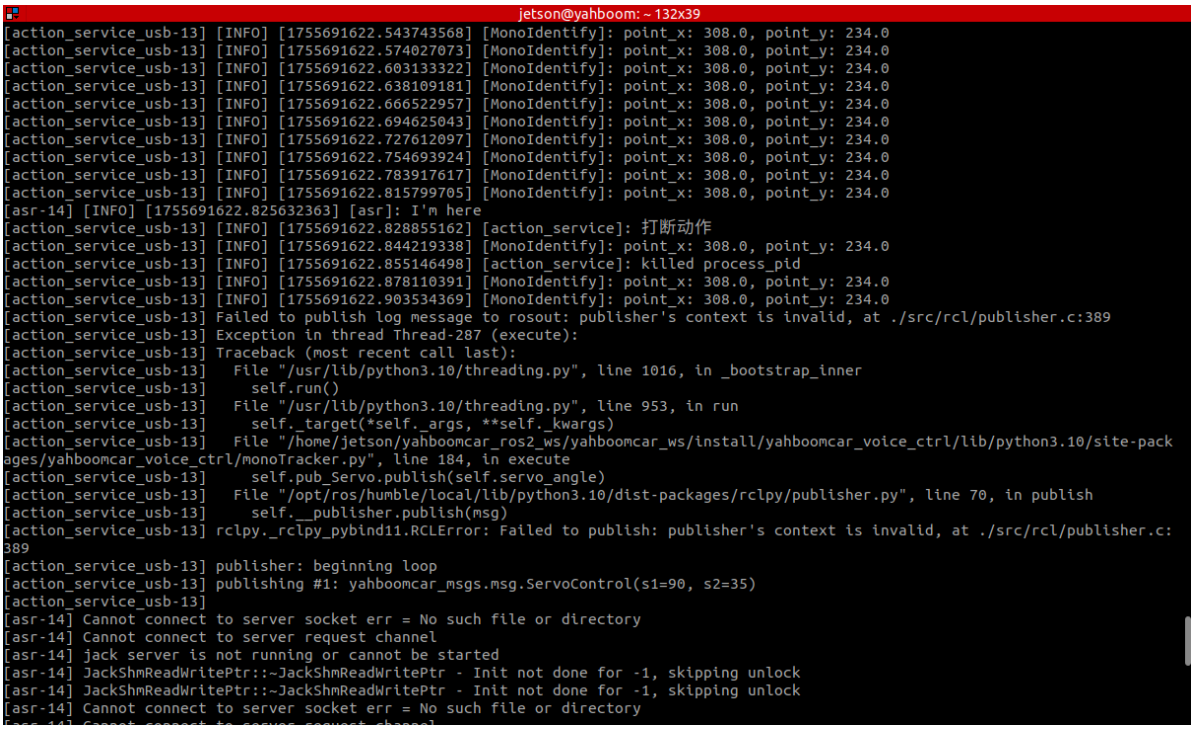


Move the object slowly, and the servo and gimbal will follow.

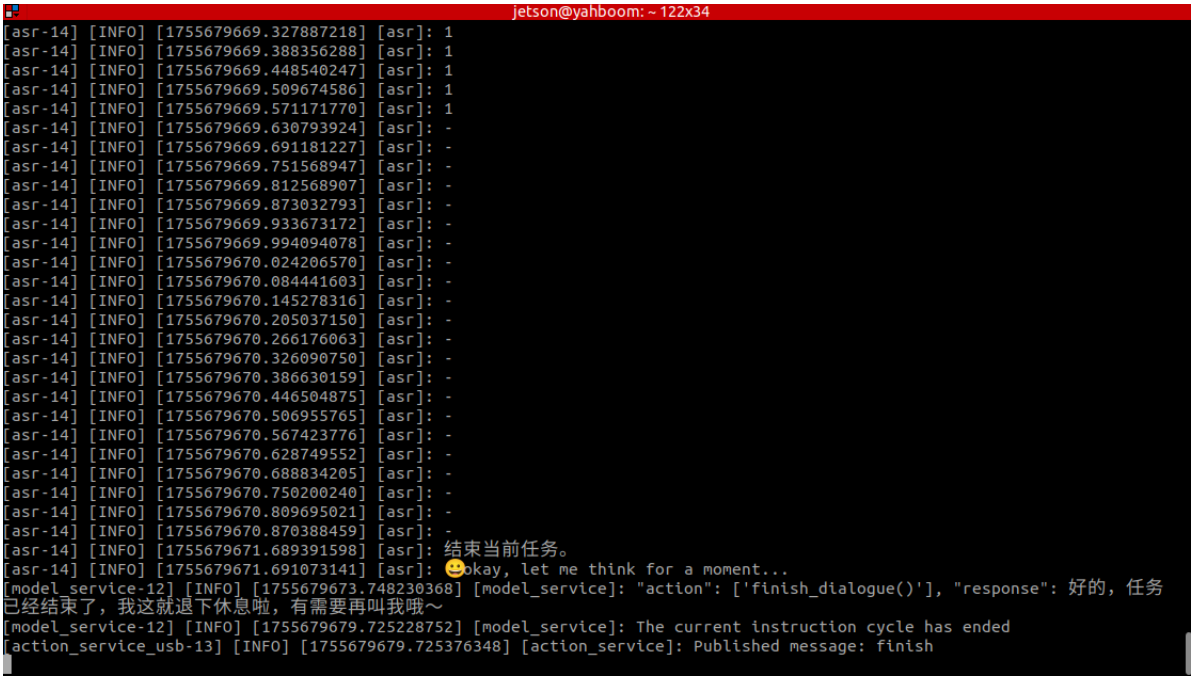
If there is no target to track in the image, the program will count down for 10 seconds, a 5-second countdown will be printed on the terminal, and the process will automatically end, indicating the task is complete. Success



To manually end a task, wake up the robot with the voice command "Hi, yahboom" The robot will respond, "I'm here, please." This will interrupt the program and automatically terminate the process, allowing you to proceed to the next command.



The robot will now enter a free conversation mode, but all conversation history will be retained. You can wake yahboom up again and select "End Current Task" to have the robot end the current task cycle, clear the conversation history, and start a new one.



4. Source Code Analysis

Source code location:

Jetson Orin Nano host:

```
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/largemodel/action_service_usb.py
```

Jetson Nano, Raspberry Pi host:

You need to enter Docker first.

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/largemodel/action_service_usb.py
```

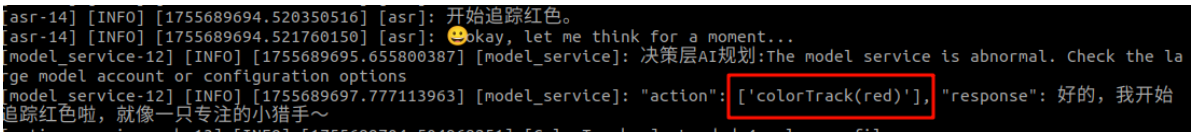
4.1 Example 1

action_service_usb.py :

Example 1 uses the **seewhat**, **color_follow**, and **stop_track** methods in the **CustomActionServer** class.

- 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'.




```
# Start the color tracking subprocess
process_1 = subprocess.Popen(['ros2', 'run', 'yahboomcar_voice_ctrl', 'colorTracker', '--ros-args', '-p', f'target_color:={target_color}'])
```

The startup program source code is located at:

`~/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 main model receives a user input command for "Stop Tracking" or "End Tracking,"

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.colorTracker_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 back to the main model at the execution layer.

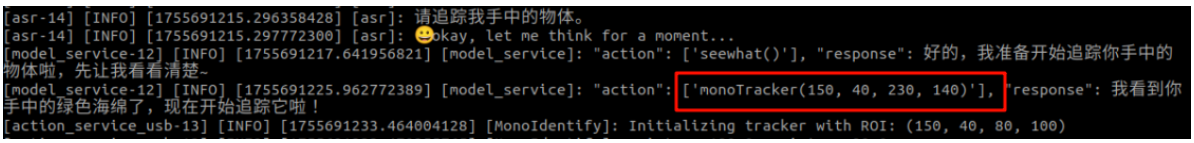
4.2 Example 2

action_service_usb.py:

Example 2 uses the **seewhat**, **monoTracker**, and **stop_track** methods in the **CustomActionServer** class.

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

The **seewhat** function primarily retrieves 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 (the upper-left corner of the image is the pixel origin). For example, the coordinates of the outer bounding box of the identified green square in Example 2 can be found from the large model's response: 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 main model receives a user input command for "Stop Tracking" or "End Tracking,"

or if 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.