

Multimodal visual understand(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 lesson uses 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 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. Running the Example

3.1 Starting the Program

For Raspberry Pi PI5 and jetson nano, you need to enter the Docker container first. For RDKX5 and Orin main controllers, this is not necessary.

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

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

```
jetson@yahboom: ~ 115x33
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 launch largemodel largemodel_control.launch.py
[INFO] [launch]: All log files can be found below /home/jetson/.ros/log/2025-08-20-15-55-46-177675-yahboom-56818
[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 [56875]
[INFO] [joint_state_publisher-2]: process started with pid [56877]
[INFO] [robot_state_publisher-3]: process started with pid [56879]
[INFO] [Ackman_driver_A1-4]: process started with pid [56881]
[INFO] [base_node_A1-5]: process started with pid [56883]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [56885]
[INFO] [ekf_node-7]: process started with pid [56887]
```

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

```
jetson@yahboom: ~ 119x33
[usb_cam_node_exe-1] [INFO] [1755676548.086026008] [usb_cam]: camera calibration URL: package:///usb_cam/config/camera_info.yaml
[usb_cam_node_exe-1] [INFO] [1755676548.165251803] [usb_cam]: Starting 'test_camera' (/dev/video0) at 640x480 via mmap (mjpeg2rgb) at 120 FPS
[usb_cam_node_exe-1] [sws_scaler @ 0xaaab21f59650] No accelerated colorspace conversion found from yuv422p to rgb24.
[usb_cam_node_exe-1] This device supports the following formats:
[usb_cam_node_exe-1] Motion-JPEG 1280 x 720 (60 Hz)
[usb_cam_node_exe-1] Motion-JPEG 1920 x 1080 (30 Hz)
[usb_cam_node_exe-1] Motion-JPEG 1024 x 768 (30 Hz)
[usb_cam_node_exe-1] Motion-JPEG 640 x 480 (120 Hz)
[usb_cam_node_exe-1] Motion-JPEG 800 x 600 (60 Hz)
[usb_cam_node_exe-1] Motion-JPEG 1280 x 1024 (30 Hz)
[usb_cam_node_exe-1] Motion-JPEG 320 x 240 (120 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 1280 x 720 (9 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 1920 x 1080 (6 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 1024 x 768 (6 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 640 x 480 (30 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 800 x 600 (20 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 1280 x 1024 (6 Hz)
[usb_cam_node_exe-1] YUVV 4:2:2 320 x 240 (30 Hz)
[joint_state_publisher-2] [INFO] [1755676548.190477321] [joint_state_publisher]: Waiting for robot_description to be published on the robot_description topic...
[usb_cam_node_exe-1] [INFO] [1755676548.970677305] [usb_cam]: Setting 'white_balance_temperature_auto' to 1
[usb_cam_node_exe-1] [INFO] [1755676548.970799444] [usb_cam]: Setting 'exposure_auto' to 1
[usb_cam_node_exe-1] [INFO] [1755676548.970816792] [usb_cam]: Setting 'exposure' to 150
[usb_cam_node_exe-1] [INFO] [1755676549.029402676] [usb_cam]: Setting 'focus_auto' to 0
[usb_cam_node_exe-1] [INFO] [1755676549.104723259] [usb_cam]: Timer triggering every 8 ms
[imu_filter_madgwick_node-6] [INFO] [1755676550.671276315] [imu_filter_madgwick]: First IMU message received.
[asr-14] [INFO] [1755676559.939338782] [asr]: The online asr model :gummy-chat-v1 is loaded
[asr-14] [INFO] [1755676559.954192471] [asr]: asr_node Initialization completed
[action_service_usb-13] [INFO] [1755676560.194134943] [action_service]: action service started...
[model_service-12] [INFO] [1755676561.034829238] [model_service]: LargeModelService node Initialization completed...
```

3.2 Test Case

- Here are two reference test cases. Users can create their own test commands.
 - Show me what items are in front of you and describe their functions.
 - Please look in front of you to see if there are blue squares and a pack of tissues. If so, nod; if not, shake your head.

3.2.1 Example 1

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

The following figure shows the Voice Active Detection (VAD):

```
jetson@yahboom: ~ 119x33
[asr-14] [INFO] [1755676559.954192471] [asr]: asr_node Initialization completed
[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 follow the instructions. The terminal will print the following information:

```
[asr-14] [INFO] [1755688248.944884941] [asr]: -
[asr-14] [INFO] [1755688249.005044626] [asr]: -
[asr-14] [INFO] [1755688250.918185262] [asr]: 帮我看你面前都有什么物品，并说一说这些物品都有什么作用。
[asr-14] [INFO] [1755688250.918896649] [asr]: 😊Okay, let me think for a moment...
[model_service-12] [INFO] [1755688252.063761592] [model_service]: 决策层AI规划:The model service is abnormal. Check the large model account or configuration options
[model_service-12] [INFO] [1755688254.435392937] [model_service]: "action": ['seewhat()'], "response": 好的，我就用我的火眼金睛看看面前都有什么物品
[model_service-12] [INFO] [1755688265.149320618] [model_service]: "action": [], "response": 我看到面前有几样东西：一个透明的玻璃瓶，看起来像是用来装水或者饮料的；旁边还有一个带金属盖的玻璃罐，可能用来储存食物或者小物件；还有一个橙色包装的纸巾盒，显然是用来擦手或清理桌面的。这些物品都挺实用的，让我感觉像在办公室里找到了生活的小确幸~
[action_service_usb-13] [INFO] [1755688291.020419771] [action_service]: Published message: 机器人反馈：回复用户完成
[model_service-12] [INFO] [1755688293.323577597] [model_service]: "action": ['finishtask()'], "response": 我已经把面前的物品都介绍完了，有需要再叫我哦
```

The presence of `finishtask()` in the action list indicates that the execution layer model has determined that the robot has completed the user's instructions and entered the waiting state. At this point, you can wake yahboom up again to complete the current task:

```
jetson@yahboom: ~ 122x34
[asr-14] [INFO] [1755679669.327887218] [asr]: 1
[asr-14] [INFO] [1755679669.388356288] [asr]: 1
[asr-14] [INFO] [1755679669.448540247] [asr]: 1
[asr-14] [INFO] [1755679669.509674586] [asr]: 1
[asr-14] [INFO] [1755679669.571171770] [asr]: 1
[asr-14] [INFO] [1755679669.630793924] [asr]: -
[asr-14] [INFO] [1755679669.691181227] [asr]: -
[asr-14] [INFO] [1755679669.751568947] [asr]: -
[asr-14] [INFO] [1755679669.812568907] [asr]: -
[asr-14] [INFO] [1755679669.873032793] [asr]: -
[asr-14] [INFO] [1755679669.933673172] [asr]: -
[asr-14] [INFO] [1755679669.994094078] [asr]: -
[asr-14] [INFO] [1755679670.024206570] [asr]: -
[asr-14] [INFO] [1755679670.084441603] [asr]: -
[asr-14] [INFO] [1755679670.145278316] [asr]: -
[asr-14] [INFO] [1755679670.205037150] [asr]: -
[asr-14] [INFO] [1755679670.266176063] [asr]: -
[asr-14] [INFO] [1755679670.326090750] [asr]: -
[asr-14] [INFO] [1755679670.386630159] [asr]: -
[asr-14] [INFO] [1755679670.446504875] [asr]: -
[asr-14] [INFO] [1755679670.506955765] [asr]: -
[asr-14] [INFO] [1755679670.567423776] [asr]: -
[asr-14] [INFO] [1755679670.628749552] [asr]: -
[asr-14] [INFO] [1755679670.688834205] [asr]: -
[asr-14] [INFO] [1755679670.750200240] [asr]: -
[asr-14] [INFO] [1755679670.809695021] [asr]: -
[asr-14] [INFO] [1755679670.870388459] [asr]: -
[asr-14] [INFO] [1755679671.689391598] [asr]: 结束当前任务。
[asr-14] [INFO] [1755679671.691073141] [asr]: 😊Okay, let me think for a moment...
[model_service-12] [INFO] [1755679673.748230368] [model_service]: "action": ['finish_dialogue()'], "response": 好的，任务已经结束了，我就退下休息啦，有需要再叫我哦~
[model_service-12] [INFO] [1755679679.725228752] [model_service]: The current instruction cycle has ended
[action_service_usb-13] [INFO] [1755679679.725376348] [action_service]: Published message: finish
```

3.2.2 Example 2

Similar to Example 1, first wake the robot with "Hi, yahboom." After the robot responds, the buzzer beeps briefly (beep—). The user can then speak. After speaking, the robot responds and moves according to the command.

Example 2 uses the gimbal servo USB camera version as an example. Nodding and shaking the head can also be used to indicate forward or reverse movement, etc.

```
[asr-14] [INFO] [1755688853.320499578] [asr]: 请你看面前有没有蓝色方块和一包纸巾，有的话你就点点头，没有的话你就摇摇头。
[asr-14] [INFO] [1755688853.322218110] [asr]: 😊Okay, let me think for a moment...
[model_service-12] [INFO] [1755688855.753276140] [model_service]: "action": ['seewhat()'], "response": 好的，我就仔细观察一下面前有没有蓝色方块和一包纸巾
[model_service-12] [INFO] [1755688865.771542685] [model_service]: "action": ['servo_nod()'], "response": 我看到了一包纸巾，但没有发现蓝色方块，不过根据你的要求，我看到纸巾就点点头啦~
[action_service_usb-13] [INFO] [1755688875.794296694] [action_service]: Published message: 机器人反馈:servo_nod()完成
[model_service-12] [INFO] [1755688879.295713993] [model_service]: "action": ['finishtask()'], "response": 我已经完成任务啦，有需要再叫我哦
[usb_cam_node_exe-1] [mjpeg @ 0xaaab1c1b7980] No JPEG data found in image
[usb_cam_node_exe-1] Failed to send AVPacket to decode: Invalid data found when processing input
```

4. Source Code Analysis

Source code located at:

Jetson Orin Nano:

```
#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
```

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

RDK X5:

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

```
/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/largemodel/model_service.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
```

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

action_service.py

In the **init_ros_communication** method of the CustomActionServer class, a topic publisher for the seewhat_handle topic is created to publish visual processing signals.

```
# Create a publisher and publish the seewhat_handle topic
self.seewhat_handle_pub = self.create_publisher(String, 'seewhat_handle', 1)
```

- USB camera, action_service_usb.py

Creates a topic subscriber for /usb_cam/image_raw to receive the camera's color image.

```
#Image topic subscriber
self.subscription = self.create_subscription(
    Image, self.image_topic, self.image_callback, 1
)
```

- nuwa depth camera, action_service_nuwa.py

Creates a topic subscriber for the /ascamera_hp60c/camera_publisher/rgb0/image command to receive color images from the depth camera.

```
#Image topic subscriber
self.subscription = self.create_subscription(
    Image, self.image_topic, self.image_callback, 1
)
```

The image_callback callback function continuously subscribes to the depth camera's color image.

When the action server calls the **seewhat** method, it publishes a signal to the seewhat_handle topic, notifying the model server's model_service node to upload an image to the multimodal large model.

```
def seewhat(self):
    self.save_single_image()
    msg = String(data="seewhat")
    self.seewhat_handle_pub.publish(
        msg
    ) # Normalize, publish to the seewhat topic, and call the large model from
    model_service.
```

At this point, the **save_single_image** method is called to save an image.png file.

Image save path:

jetson Orin Nano :

```
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/install/largemodel/share/largemode
l/resources_file/image.png
```

For the Jetson Nano and Raspberry Pi hosts, you must first enter Docker:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/install/largemodel/share/largemodel/resou
rces_file/image.png
```

```
def image_callback(self, msg):#Image callback function
    self.image_msg = msg

def save_single_image(self):
    """
    保存一张图片 / Save a single image
    """
    if self.image_msg is None:
        self.get_logger().warning("No image received yet.") # 尚未接收到图像...
        return
    try:
        # 将ROS图像消息转换为OpenCV图像 / Convert ROS image message to OpenCV image
        cv_image = self.bridge.imgmsg_to_cv2(self.image_msg, "bgr8")
        # 保存图片 / Save the image
        cv2.imwrite(self.image_save_path, cv_image)
        display_thread = threading.Thread(target=self.display_saved_image)
        display_thread.start()
```

```

except Exception as e:
    self.get_logger().error(f"Error saving image: {e}") # Error saving
image...

```

model_service.py

In the **init_ros_communication** method of the LargeModelService class,
a subscriber to the topic "seewhat_handle" is created to receive visual processing signals.

```

#Create a Seewhat subscriber
self.seewhat_sub = self.create_subscription(String, 'seewhat_handle',
self.seewhat_callback, 1)

```

When the seewhat_callback callback function receives the visual processing signal, it calls the **dual_large_model_mode** method, and then calls the **instruction_process** method to request inference and upload an image from the depth camera to the multimodal large model.

```

def seewhat_callback(self, msg):
    if msg.data == "seewhat":
        if (
            self.regional_setting == "China"
        ): # 在线模型推理方式: 决策层推理+执行层监督 / Online model inference method:
Decision layer reasoning + Execution layer supervision
            self.dual_large_model_mode(type="image")
        else:
            self.dual_large_model_international_model(type="image")

```

```

def instruction_process(self, prompt, type, conversation_id=None):
    """
    根据输入信息的类型（文字/图片），构建不同的请求体进行推理，并返回结果）
    Based on the type of input information (text/image), construct different
    request bodies for inference and return the result.
    """
    json_str = None
    prompt_seewhat = self.prompt_dict[self.language].get("prompt_2")
    if self.regional_setting == "China":
        if type == "text":
            raw_content = self.model_client.multimodalinfer(prompt)
        elif type == "image":
            raw_content = self.model_client.multimodalinfer(
                prompt_seewhat, image_path=self.image_save_path
            )
        json_str = self.extract_json_content(raw_content)
    elif self.regional_setting == "international":
        if type == "text":
            result = self.model_client.TaskExecution(
                input=prompt,
                map_mapping=self.map_mapping,
                language=self.language_dict[self.language],
                conversation_id=conversation_id,
            )
        if result[0]:
            json_str = self.extract_json_content(result[1])
            self.conversation_id = result[2]

```

```
        else:
            self.get_logger().info(f"ERROR:{result[1]}")
    elif type == "image":
        result = self.model_client.TaskExecution(
            input=prompt_seewhat,
            map_mapping=self.map_mapping,
            language=self.language_dict[self.language],
            image_path=self.image_save_path,
            conversation_id=conversation_id,
        )
        if result[0]:
            json_str = self.extract_json_content(result[1])
            self.conversation_id = result[2]
        else:
            self.get_logger().info(f"ERROR:{result[1]}")
```