

Multimodal visual understand

1. Course Content

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

2. Preparation

2.1 Content Description

This course uses the Jetson Orin NX 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 the **[Configuration and Operation Guide] -- [Enter the Docker (Jetson Nano and Raspberry Pi 5 users see here)]** section of this product tutorial. For Orin and NX boards, simply open a terminal and enter the commands mentioned in this course.

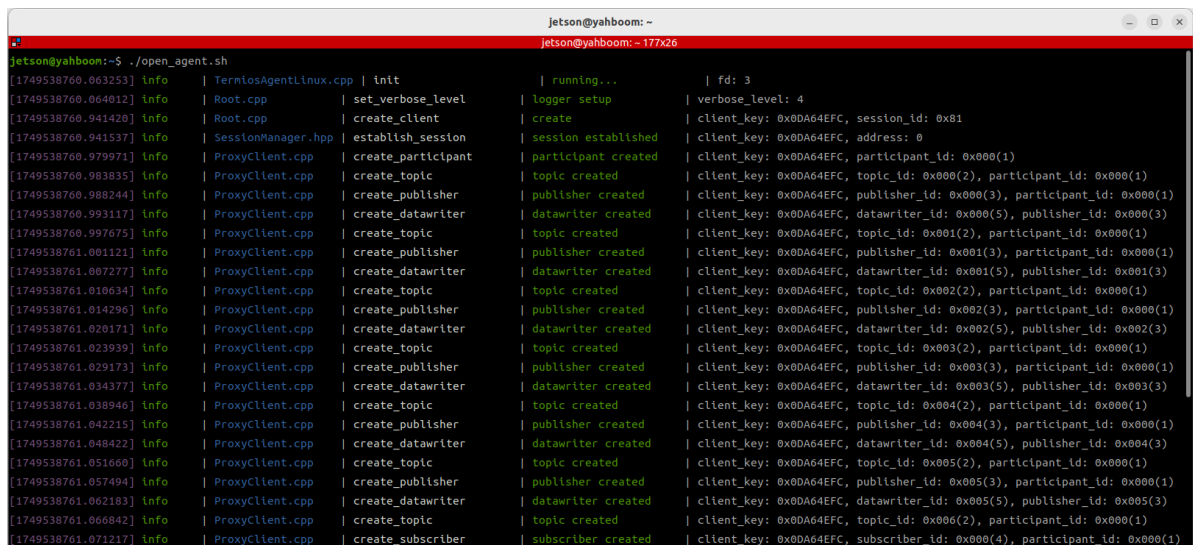
2.2 Start the agent

Note: To test all cases, you must first start the docker agent. If it has already been started, you do not need to start it again

Enter the command in the vehicle terminal:

```
sh start_agent.sh
```

The terminal prints the following information, indicating that the connection is successful



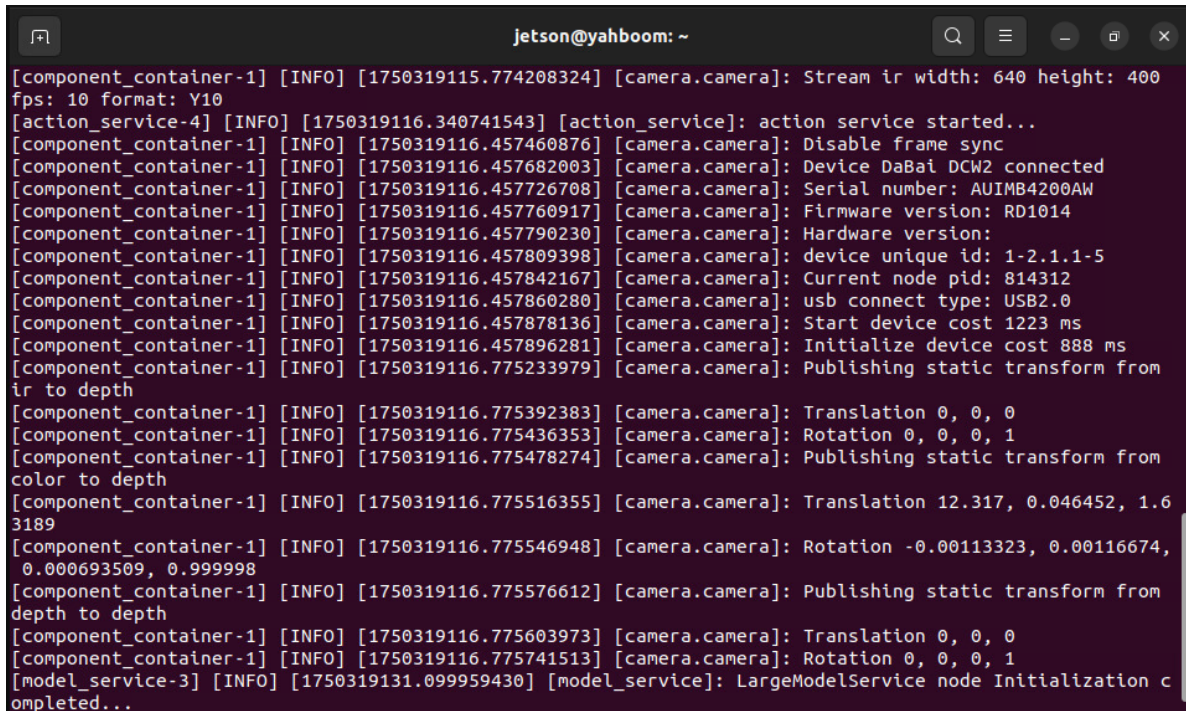
```
Jetson@yahboom: ~  
Jetson@yahboom: ~ 177x26  
Jetson@yahboom:~$ ./open_agent.sh  
[1749538760.063253] Info | TermiosAgentLinux.cpp | init | running... | fd: 3  
[1749538760.064012] Info | Root.cpp | set_verbose_level | logger setup | verbose_level: 4  
[1749538760.941420] Info | Root.cpp | create_client | create | client_key: 0x0DA64EFC, session_id: 0x81  
[1749538760.941537] Info | SessionManager.hpp | establish_session | session established | client_key: 0x0DA64EFC, address: 0  
[1749538760.979971] Info | ProxyClient.cpp | create_participant | participant created | client_key: 0x0DA64EFC, participant_id: 0x000(1)  
[1749538760.983835] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x00(2), participant_id: 0x000(1)  
[1749538760.988244] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x000(3), participant_id: 0x000(1)  
[1749538760.993117] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x00(5), publisher_id: 0x000(3)  
[1749538760.997675] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x001(2), participant_id: 0x000(1)  
[1749538761.001121] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x001(3), participant_id: 0x000(1)  
[1749538761.007277] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x001(5), publisher_id: 0x001(3)  
[1749538761.010634] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x002(2), participant_id: 0x000(1)  
[1749538761.014296] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x002(3), participant_id: 0x000(1)  
[1749538761.020171] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x002(5), publisher_id: 0x002(3)  
[1749538761.023939] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x003(2), participant_id: 0x000(1)  
[1749538761.029173] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x003(3), participant_id: 0x000(1)  
[1749538761.034377] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x003(5), publisher_id: 0x003(3)  
[1749538761.038946] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x004(2), participant_id: 0x000(1)  
[1749538761.042215] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x004(3), participant_id: 0x000(1)  
[1749538761.048422] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x004(5), publisher_id: 0x004(3)  
[1749538761.051660] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x005(2), participant_id: 0x000(1)  
[1749538761.057494] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x005(3), participant_id: 0x000(1)  
[1749538761.062183] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x005(5), publisher_id: 0x005(3)  
[1749538761.066842] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x006(2), participant_id: 0x000(1)  
[1749538761.071217] Info | ProxyClient.cpp | create_subscriber | subscriber created | client_key: 0x0DA64EFC, subscriber_id: 0x000(4), participant_id: 0x000(1)
```

3. Run the case

3.1 Start the program

Open the terminal on the vehicle and enter the command:

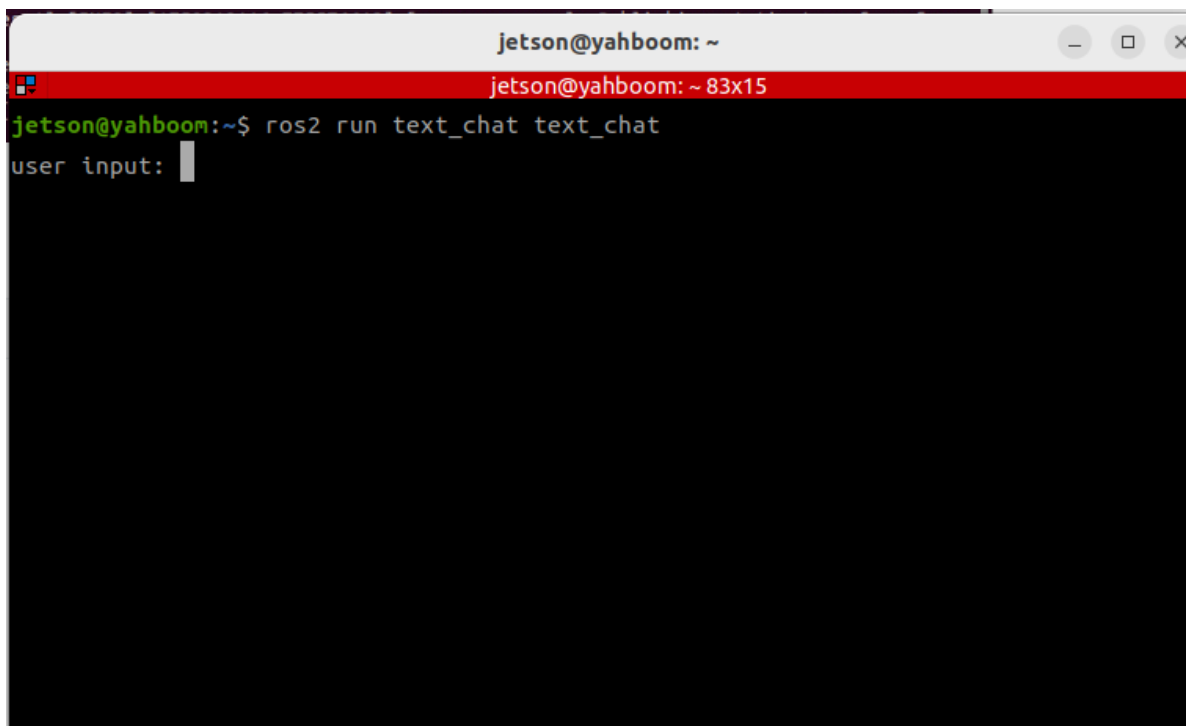
```
ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
```



```
jetson@yahboom: ~  
[component_container-1] [INFO] [1750319115.774208324] [camera.camera]: Stream ir width: 640 height: 400  
fps: 10 format: Y10  
[action_service-4] [INFO] [1750319116.340741543] [action_service]: action service started...  
[component_container-1] [INFO] [1750319116.457460876] [camera.camera]: Disable frame sync  
[component_container-1] [INFO] [1750319116.457682003] [camera.camera]: Device DaBai DCW2 connected  
[component_container-1] [INFO] [1750319116.457726708] [camera.camera]: Serial number: AUIMB4200AW  
[component_container-1] [INFO] [1750319116.457760917] [camera.camera]: Firmware version: RD1014  
[component_container-1] [INFO] [1750319116.457790230] [camera.camera]: Hardware version:  
[component_container-1] [INFO] [1750319116.457809398] [camera.camera]: device unique id: 1-2.1.1-5  
[component_container-1] [INFO] [1750319116.457842167] [camera.camera]: Current node pid: 814312  
[component_container-1] [INFO] [1750319116.457860280] [camera.camera]: usb connect type: USB2.0  
[component_container-1] [INFO] [1750319116.457878136] [camera.camera]: Start device cost 1223 ms  
[component_container-1] [INFO] [1750319116.457896281] [camera.camera]: Initialize device cost 888 ms  
[component_container-1] [INFO] [1750319116.775233979] [camera.camera]: Publishing static transform from  
ir to depth  
[component_container-1] [INFO] [1750319116.775392383] [camera.camera]: Translation 0, 0, 0  
[component_container-1] [INFO] [1750319116.775436353] [camera.camera]: Rotation 0, 0, 0, 1  
[component_container-1] [INFO] [1750319116.775478274] [camera.camera]: Publishing static transform from  
color to depth  
[component_container-1] [INFO] [1750319116.775516355] [camera.camera]: Translation 12.317, 0.046452, 1.6  
3189  
[component_container-1] [INFO] [1750319116.775546948] [camera.camera]: Rotation -0.00113323, 0.00116674,  
0.000693509, 0.999998  
[component_container-1] [INFO] [1750319116.775576612] [camera.camera]: Publishing static transform from  
depth to depth  
[component_container-1] [INFO] [1750319116.775603973] [camera.camera]: Translation 0, 0, 0  
[component_container-1] [INFO] [1750319116.775741513] [camera.camera]: Rotation 0, 0, 0, 1  
[model_service-3] [INFO] [1750319131.099959430] [model_service]: LargeModelService node Initialization c  
ompleted...
```

Take the virtual machine as an example, open a terminal and start

```
ros2 run text_chat text_chat
```



```
jetson@yahboom: ~  
jetson@yahboom: ~ 83x15  
jetson@yahboom:~$ ros2 run text_chat text_chat  
user input: 
```

3.2 Test Cases

Here are two reference test cases, users can compile their own test instructions

- Please show me what items are in front of you and tell me what they are used for.
- Please look in front of you to see if there is a blue cube and a pack of tissues. If yes, nod your head. If not, shake your head.

3.2.1 Case 1

Enter the test case in the text interactive terminal, wait for the model to think, and then the model will respond to the user and perform actions according to the user's instructions.

```
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: Please show me what items are in front of you and tell me what they are used for.
okay@, let me think for a moment... -[INFO] [1755158911.308052374] [text_chat_node]: "action": ['seewhat()'], "response": Sure! I'm going to take a quick look around and see what's in front of me. Just a sec!
user input: [INFO] [1755158918.419814638] [text_chat_node]: "action": [], "response": Alright, I've got a clear view of what's in front of me! There's a small fan on the desk, probably here to keep things cool during long work sessions. Next to it, I see a power strip with several chargers plugged in—perfect for keeping all your devices powered up. There's also a hand sanitizer bottle, which is great for staying clean and germ-free. A pen holder with pens and markers is nearby, so someone must be into writing or drawing. And of course, there are cables everywhere—typical office chaos! Let me know if you want me to focus on anything specific!
[INFO] [1755158923.159358078] [text_chat_node]: "action": ['finishtask()'], "response": Got it! I've shared everything I could see and their uses. Let me know if you'd like to explore something else!
End current task
okay@, let me think for a moment... \[INFO] [1755158935.086500530] [text_chat_node]: "action": ['finish_dialogue()'], "response": Alright, I'm stepping down for now. Have a wonderful day, and let me know if you need help later!
user input: █
```

After completing a task, the robot enters a waiting state. You can continue the conversation at this point. The commands you enter are directly passed 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.

3.2.2 Case 2

The method is the same as in Case 1. Enter Case 2 in the terminal, and the model will respond and perform actions according to the instructions.

```
user input: Please look in front of you to see if there is a blue cube and a pack of tissues. If yes, nod your head. If not, shake your head.
okay@, let me think for a moment... -[INFO] [1755158997.734682684] [text_chat_node]: Decision making AI planning:1. Observe the current frame.
2. If both the blue cube and the pack of tissues are in the frame, call the function for the robotic arm to shake its head.
3. If either the blue cube or the pack of tissues is not in the frame, call the function for the robotic arm to shake its head.
user input: [INFO] [1755159000.891499449] [text_chat_node]: "action": ['seewhat()', 'arm_nod()'], "response": I'm checking what's in front of me right now, and if I see both the blue cube and the tissues, I'll give a little nod. If not, I'll shake my head to let you know!
[INFO] [1755159006.163648461] [text_chat_node]: "action": ['arm_shake()'], "response": Hmm, I looked around carefully, but I couldn't spot the blue cube or the pack of tissues. Guess they're hiding somewhere else!
[INFO] [1755159010.678553180] [text_chat_node]: "action": ['arm_shake()'], "response": I just checked again, and I still don't see the blue cube or the tissues. Maybe they're not here at all? I'm shaking my head to let you know!
[INFO] [1755159017.282908564] [text_chat_node]: "action": ['finishtask()'], "response": Alright, I've checked and shaken my head—no blue cube or tissues in sight. That's all for now!
[INFO] [1755159020.550603405] [text_chat_node]: "action": ['finishtask()'], "response": I've already checked and confirmed—no blue cube or tissues here. I'm all done for now!
End current task
okay@, let me think for a moment... -[INFO] [1755159059.809154571] [text_chat_node]: "action": ['finish_dialogue()'], "response": Got it! I'm stepping down and wrapping up this task. Have a great day!
user input: █
```

4. Source code analysis

The source code is located at:

jetson orin nano, jetson orin NX host:

```
/home/jetson/M3Pro_ws/src/largemodel/largemodel/action_service.py
```

```
/home/jetson/M3Pro_ws/src/largemodel/largemodel/model_service.py
```

Jetson Nano, Raspberry Pi host:

You need to enter docker first

```
/home/jetson/M3Pro_ws/src/largemodel/largemodel/action_service.py
```

```
root/M3Pro_ws/src/largemodel/largemodel/model_service.py
```

action_service.py program:

In the `init_ros_comunication` method of the `CustomActionServer` class, a topic publisher called `seewhat_handle` 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)
```

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

```
def seewhat(self):
    msg = String(data='seewhat')
    self.seewhat_handle_pub.publish(msg) # Normalize, publish the seewhat topic,
    and let the master node call the large model
```

model_service.py program:

In the `init_ros_comunication` method of the `LargeModelService` class

Created a `seewhat_handle` topic subscriber to receive visual processing signals

Created a `camera/color/image_raw` topic subscriber to receive the color image from the depth camera

```
# Create a seewhat subscriber
self.seewhat_sub = self.create_subscription(String, 'seewhat_handle',
self.seewhat_callback, 1)
# ImageTopicSubscribers
self.subscription = self.create_subscription(Image, self.image_topic,
self.image_callback, 1)
```

When the `seewhat_callback` callback function receives the visual processing signal, it calls the `dual_large_model_mode` method to first save an image from the depth camera and then upload it to the multimodal large model.

```
def seewhat_callback(self, msg):
    if msg.data == "seewhat":
        if self.use_double_llm: # Online model inference method: decision layer
            reasoning + execution layer supervision
            self.dual_large_model_mode(type="image")
        else: # Online model inference method: decision layer reasoning +
            execution layer supervision
            self.single_largeModel_infer(image_path=self.image_save_path)
```

The image_callback callback function will continue to subscribe to the color image of the depth camera. When the image needs to be saved, the save_single_image method will be called to save an image.png image file.

Image save path:

jetson orin nano, jetson orin NX host:

```
/home/jetson/M3Pro_ws/install/largemodel/share/largemodel/resources_file/image.png
```

Jetson Nano and Raspberry Pi host need to enter docker first:

```
root/M3Pro_ws/install/largemodel/share/largemodel/resources_file/image.png
```

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

def save_single_image(self):
    if self.image_msg is None:
        self.get_logger().warning('No image received yet.')
        return
    try:
        # Convert ROS image message to OpenCV image
        cv_image = self.bridge.imgmsg_to_cv2(self.image_msg, 'bgr8')
        # Save the image
        cv2.imwrite(self.image_save_path, cv_image)
        # self.get_logger().info(f'Saved image: {self.image_save_path}')
    except Exception as e:
        self.get_logger().error(f'Error saving image: {e}')
```