

Robotic arm gripping + multimodal visual understanding + SLAM navigation

1. Course Content

1. Learn robot SLAM navigation, robotic arm gripping, and AI large model visual understanding composite function cases
2. Study the new key source code in the tutorial
3. **Note: This section requires you to first configure the map mapping file according to the previous section Multimodal Visual Understanding + SLAM Navigation**

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.

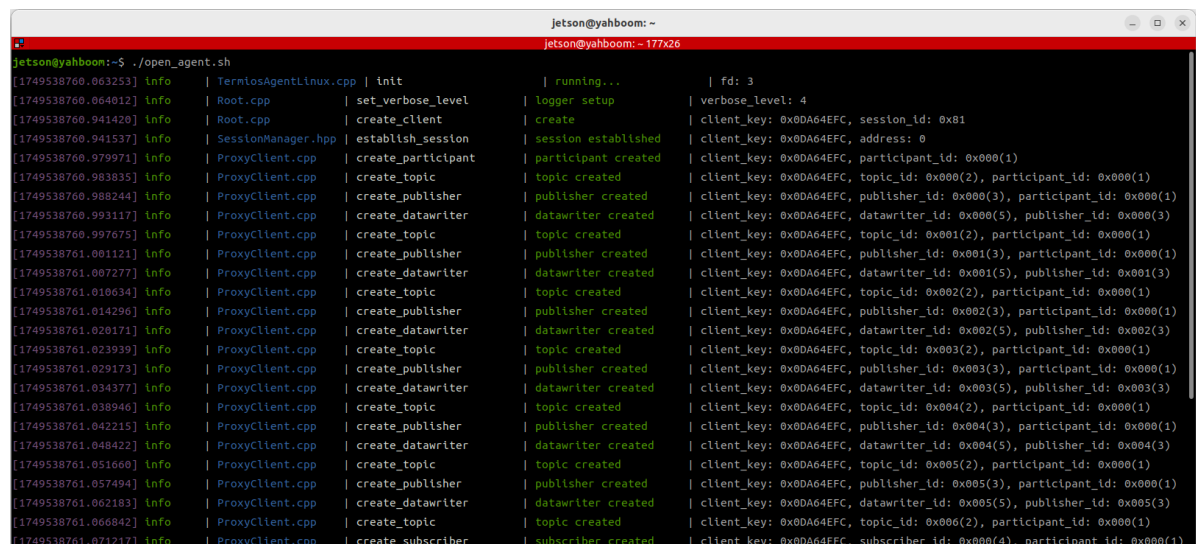
2.2 Start the Agent

Note: To test all cases, you must start the docker agent first. 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: 0x000(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: 0x000(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)  
[1749538761.051600] 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)  
[1749538761.066042] 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)
```

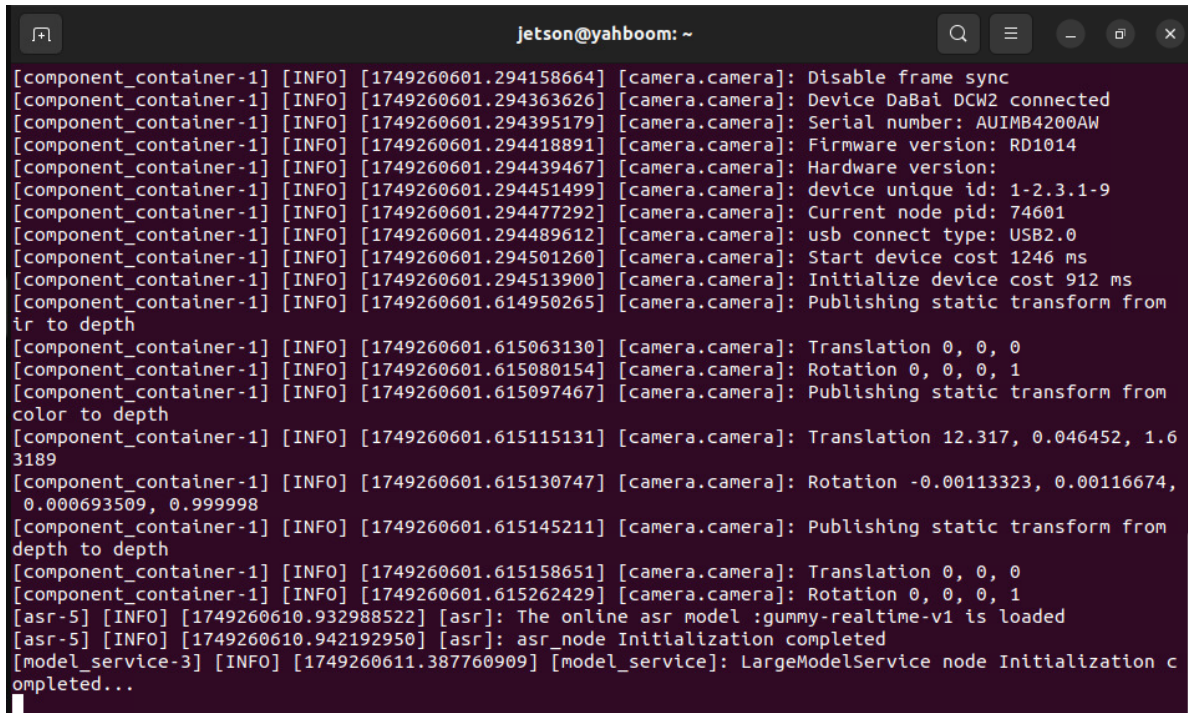
3. Run the case

3.1 Startup Program

Connect to the robot desktop via VNC, open a terminal, and start the command

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

After initialization is complete, the following content will be displayed



```
jetson@yahboom: ~  
[component_container-1] [INFO] [1749260601.294158664] [camera.camera]: Disable frame sync  
[component_container-1] [INFO] [1749260601.294363626] [camera.camera]: Device DaBai DCW2 connected  
[component_container-1] [INFO] [1749260601.294395179] [camera.camera]: Serial number: AUIMB4200AW  
[component_container-1] [INFO] [1749260601.294418891] [camera.camera]: Firmware version: RD1014  
[component_container-1] [INFO] [1749260601.294439467] [camera.camera]: Hardware version:  
[component_container-1] [INFO] [1749260601.294451499] [camera.camera]: device unique id: 1-2.3.1-9  
[component_container-1] [INFO] [1749260601.294477292] [camera.camera]: Current node pid: 74601  
[component_container-1] [INFO] [1749260601.294489612] [camera.camera]: usb connect type: USB2.0  
[component_container-1] [INFO] [1749260601.294501260] [camera.camera]: Start device cost 1246 ms  
[component_container-1] [INFO] [1749260601.294513900] [camera.camera]: Initialize device cost 912 ms  
[component_container-1] [INFO] [1749260601.614950265] [camera.camera]: Publishing static transform from  
ir to depth  
[component_container-1] [INFO] [1749260601.615063130] [camera.camera]: Translation 0, 0, 0  
[component_container-1] [INFO] [1749260601.615080154] [camera.camera]: Rotation 0, 0, 0, 1  
[component_container-1] [INFO] [1749260601.615097467] [camera.camera]: Publishing static transform from  
color to depth  
[component_container-1] [INFO] [1749260601.615115131] [camera.camera]: Translation 12.317, 0.046452, 1.6  
3189  
[component_container-1] [INFO] [1749260601.615130747] [camera.camera]: Rotation -0.00113323, 0.00116674,  
0.000693509, 0.999998  
[component_container-1] [INFO] [1749260601.615145211] [camera.camera]: Publishing static transform from  
depth to depth  
[component_container-1] [INFO] [1749260601.615158651] [camera.camera]: Translation 0, 0, 0  
[component_container-1] [INFO] [1749260601.615262429] [camera.camera]: Rotation 0, 0, 0, 1  
[asr-5] [INFO] [1749260610.932988522] [asr]: The online asr model :gummy-realtime-v1 is loaded  
[asr-5] [INFO] [1749260610.942192950] [asr]: asr_node Initialization completed  
[model_service-3] [INFO] [1749260611.387760909] [model_service]: LargeModelService node Initialization c  
ompleted...
```

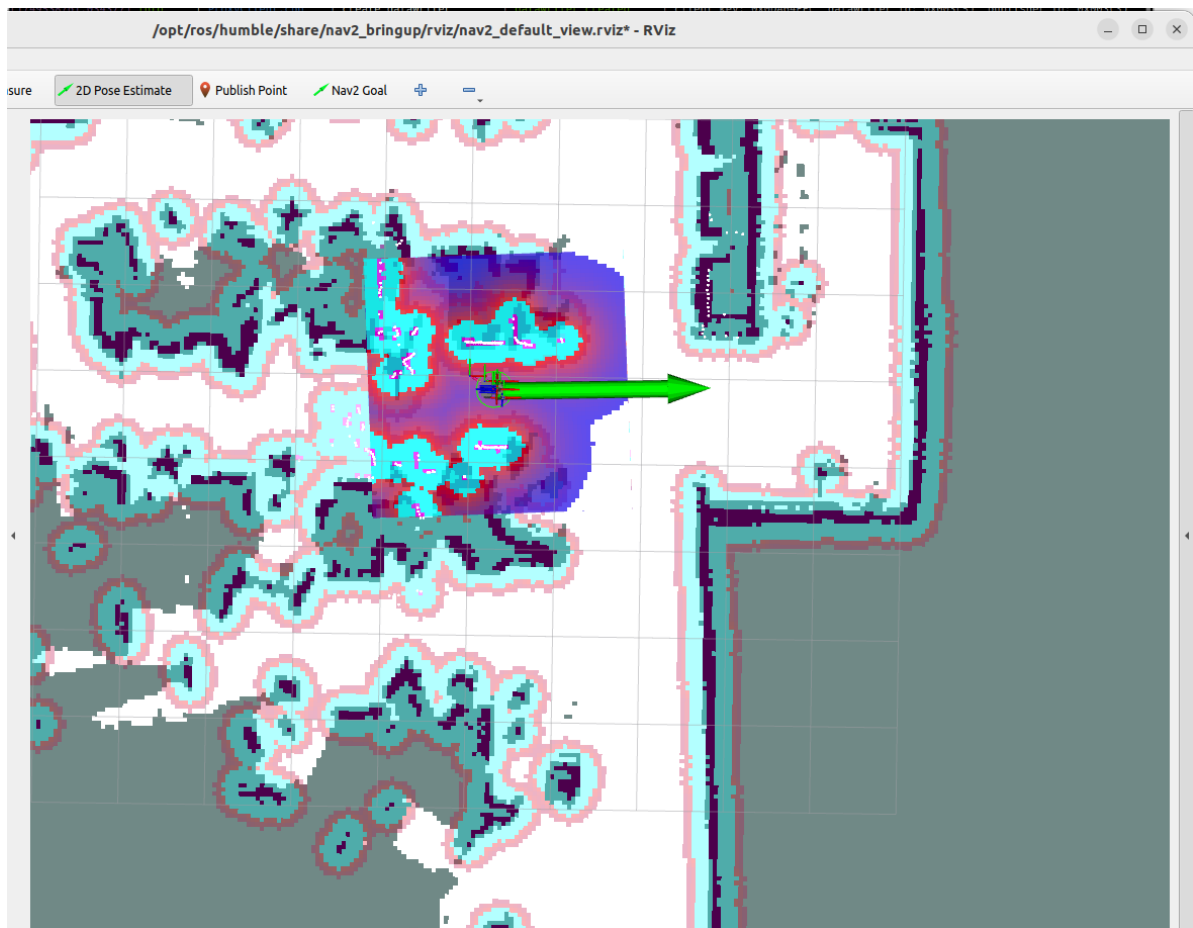
Open two terminals on the vehicle and enter the following commands:

```
ros2 launch M3Pro_navigation base_bringup.launch.py  
ros2 launch M3Pro_navigation navigation2.launch.py
```

Create a new terminal on the virtual terminal and start

```
ros2 launch slam_view nav_rviz.launch.py
```

Then follow the process of starting the navigation function to initialize the positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to enter the selection state. Roughly mark the position and orientation of the robot on the map. After initialization positioning, the preparation work is completed.



3.2 Test Cases

Here are some test cases for reference. Users can write their own dialogue commands.

- I am in the master bedroom now. Please help me take the red block in front of you to the master bedroom.

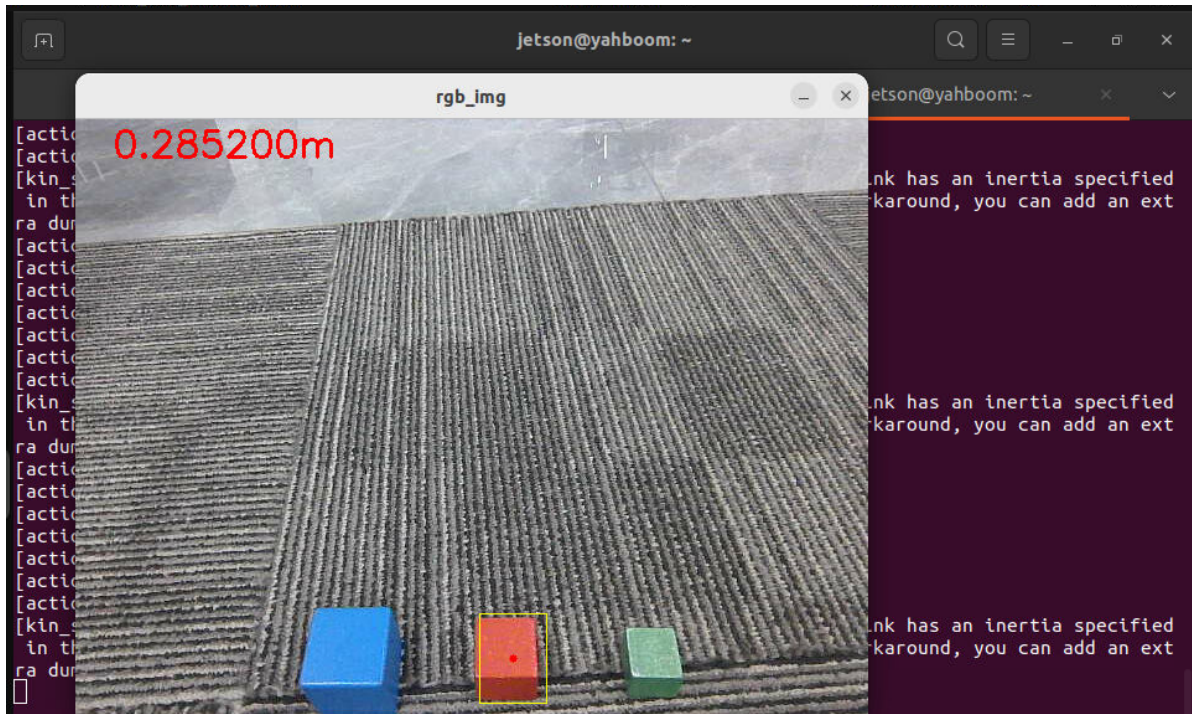
First, use "Hi, yahboom" to wake up the robot. The robot responds: "I'm here, please tell me what to do." After the robot answers, the buzzer beeps briefly (beep—), and the user can speak. After the robot replies, the terminal prints the following information:

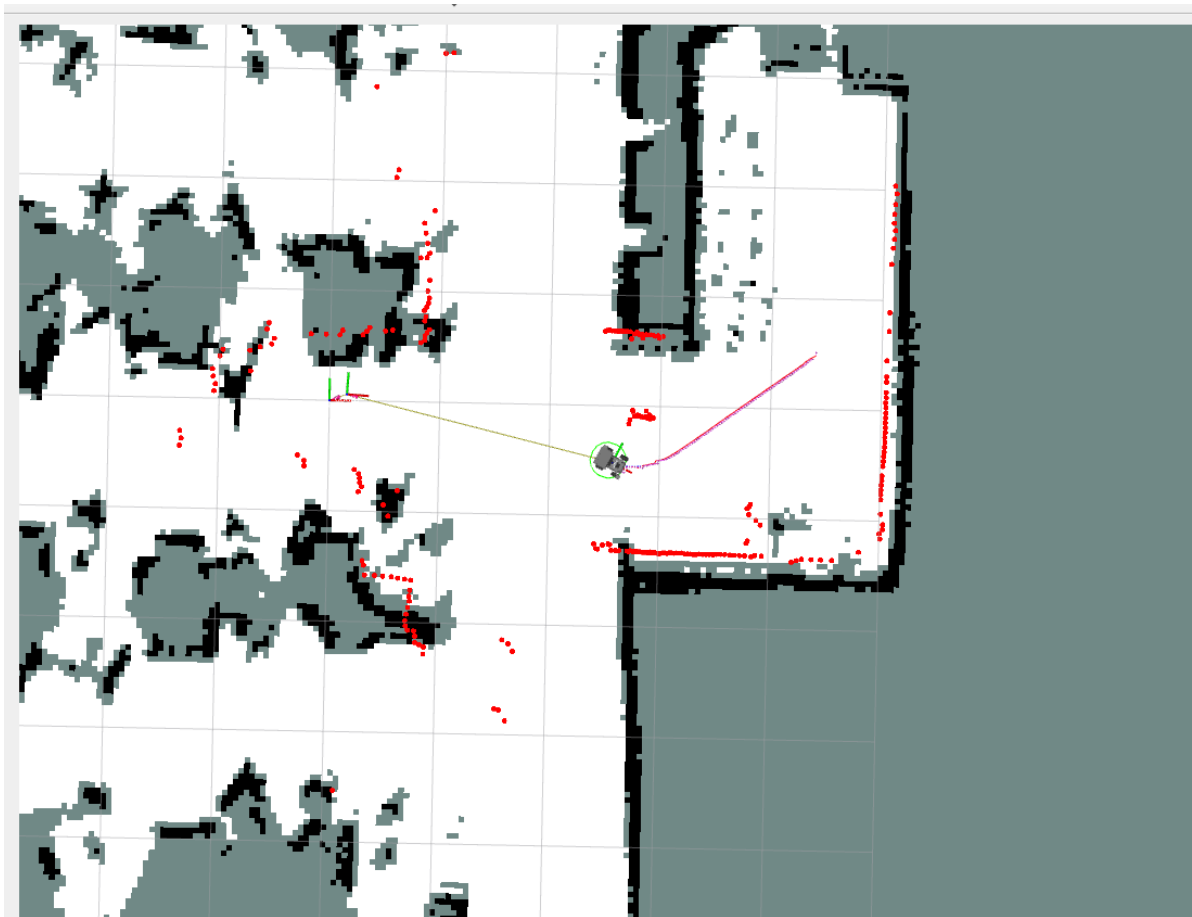

```
jetson@yahboom: ~  
[asr-5] [INFO] [1749549228.396757797] [asr]: 0  
[asr-5] [INFO] [1749549228.486669892] [asr]: 0  
[asr-5] [INFO] [1749549228.578750722] [asr]: 0  
[asr-5] [INFO] [1749549228.670072394] [asr]: 0  
[asr-5] [INFO] [1749549228.762238058] [asr]: 0  
[asr-5] [INFO] [1749549228.852788924] [asr]: 0  
[asr-5] [INFO] [1749549228.944713333] [asr]: 0  
[asr-5] [INFO] [1749549229.035328393] [asr]: 0  
[asr-5] [INFO] [1749549229.125801048] [asr]: 0  
[asr-5] [INFO] [1749549229.217032830] [asr]: 0  
[asr-5] [INFO] [1749549229.307577808] [asr]: 0  
[asr-5] [INFO] [1749549231.865243587] [asr]: 我现在在主卧室，请你帮我把你面前的红色方块拿到主卧。  
[asr-5] [INFO] [1749549231.866657804] [asr]: okay😊, let me think for a moment...  
[model_service-3] [INFO] [1749549236.016431076] [model_service]: The upcoming task to be carried out: 1.  
调用视觉函数获取红色方块在画面中的坐标  
[model_service-3] 2. 使用机械臂夹取红色方块  
[model_service-3] 3. 导航前往主卧室  
[model_service-3] 4. 放下红色方块  
[model_service-3] [INFO] [1749549238.247663068] [model_service]: "action": ['seewhat()'], "response": 好的，我先看看面前的红色方块在哪里。  
[model_service-3] [INFO] [1749549249.887211674] [model_service]: "action": ['grasp_obj(347,280,396,341)'], "response": 我已经看到了红色方块的位置，现在准备用机械臂夹取它。  
[kin_srv-2] [WARN] [1749549257.735376647] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.  
[action_service-4] start  
[action_service-4] [INFO] [1749549261.725444770] [image_converter]: Received: [347 280 396 341]
```

The decision-making model planning task steps are:

```
[model_service-3] [INFO] [1749549236.016431076] [model_service]: The upcoming task to be carried out: 1.  
调用视觉函数获取红色方块在画面中的坐标  
[model_service-3] 2. 使用机械臂夹取红色方块  
[model_service-3] 3. 导航前往主卧室  
[model_service-3] 4. 放下红色方块  
[model_service-3] [INFO] [1749549238.247663068] [model_service]: "action": ['seewhat()'], "response": 好的，我先看看面前的红色方块在哪里。  
[model_service-3] [INFO] [1749549249.887211674] [model_service]: "action": ['grasp_obj(347,280,396,341)'], "response": 我已经看到了红色方块的位置，现在准备用机械臂夹取它。  
[kin_srv-2] [WARN] [1749549257.735376647] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.  
[action_service-4] start  
[action_service-4] [INFO] [1749549261.725444770] [image_converter]: Received: [347 280 396 341]
```

According to the task steps planned by the decision-making model, the robot will first observe the environment in front of it, then grab the red square, and then navigate to the target point according to the path planned by the global planner.





After arriving at the navigation target point, the robot will use its robotic arm to put down the red block in its hand and prompt the user that the task is completed. At this time, the robot enters a waiting state, and the user can choose to continue the conversation or command the robot to end the current task and start a new task cycle.

```

jetson@yahboom: ~
jetson@yahboom: ~
jetson@yahboom: ~

in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[kin_srv-2] [WARN] [1749554694.230892821] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[kin_srv-2] [WARN] [1749554694.236474280] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[kin_srv-2] [WARN] [1749554694.240666463] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[kin_srv-2] [WARN] [1749554694.244799797] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[kin_srv-2] [WARN] [1749554694.250654158] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[model_service-3] [INFO] [1749554702.868951526] [model_service]: "action": ['navigation(A)'], "response": 红色方块已经夹取成功, 现在我将前往主卧室。
[action_service-4] [INFO] [1749554708.400056960] [action_service]: navigation Goal accepted
[action_service-4] [INFO] [1749554771.641868476] [action_service]: Navigation succeeded!
[model_service-3] [INFO] [1749554775.480216042] [model_service]: "action": ['putdown()'], "response": 我已经到达主卧室, 现在将红色方块放下。
[model_service-3] [INFO] [1749554791.990187985] [model_service]: "action": ['finishtask()'], "response": 红色方块已经成功放在主卧室了, 任务完成!

```

4. Source code analysis

The source code is located at:

jetson orin nano, jetson orin NX host:

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

Jetson Nano, Raspberry Pi host, you need to enter Docker first:

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

action_service.py program:

The case uses **the seewhat , navigation , load_target_points , putdown , and grasp_obj** methods in **the CustomActionServer class**. **Seewhat , navigation , load_target_points , and grasp_obj** have been explained in the previous chapters [2. Multimodal Visual Understanding, 3. Multimodal Visual Understanding + Robotic Arm Grabbing, and 4. Multimodal Visual Understanding + SLAM Navigation]. Here we explain the newly appeared **putdown function**.

The putdown function controls the robotic arm to put down the object it has gripped. After gripping an object, the robotic arm will be in a gripping state. Calling this function allows the robotic arm to put down the gripped object. The principle is to control the robotic arm posture by publishing the robotic arm joint topic.

```
def putdown ( self ):
    self . pubSix_Arm ( self . putsown_joints ) # Lower the robotic arm
    time . sleep ( 3 )
    self . pubSingle_Arm ( 6 , 30 , 2000 ) #The robotic arm opens the grip and
    puts down the item
    time . sleep ( 3 )
    self . pubSix_Arm ( self . init_joints ) #Robotic arm retracts
    self . action_status_pub ( f'Robot feedback: Execution of putdown()
    completed' )
```