

Intention Estimation + Multimodal Visual Understand + SLAM Navigation + Visual Functions (Text Version)

[Intention Estimation + Multimodal Visual Understand + SLAM Navigation + Visual Functions \(Text Version\)](#)

1. Course Content
2. Preparation
 - 2.1 Content Description
3. Document Configuration
 - 3.1 Create a md document
 - 3.2 Upload to the DIFY backend
4. Run the Example
 - 4.1 Starting the Program
 - 4.1.1 Jetson Nano Board Startup Steps:
 - 4.1.2 Other main control startup steps:
 - 4.2 Test Cases
 - 4.2.1 Example 1
5. Source Code Analysis
 - 5.1 Example 1

1. Course Content

Note: This section requires you to first complete the map file configuration as described in the [Multimodal Visual Understand + SLAM Navigation] section.

1. Learning to Use the RAG Knowledge Base to Train Personal Intention Understanding

Course Review:

The RAG knowledge base can be used to expand the knowledge base of the large model. This section explains how to expand the RAG knowledge base so that the large model can understand personal intentions.

Important Note! ! :

1. The expanded personal intention understanding function may vary from user to user, and the large model's responses are divergent, so the actual debugging results may not be exactly the same.
2. The expanded personal intention understanding function must comply with social norms and laws and regulations. Technical Support assumes no responsibility for the final debugging results or any impact of this course.

2. Preparation

2.1 Content Description

This lesson uses the Raspberry Pi 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"`

 For the same test command, the model's responses may not be exactly the same and may differ slightly from the screenshots in the tutorial. To increase or decrease the diversity of the model's responses, refer to the section on configuring the decision-making model parameters in **[03.AI Model Basics] -- [5.Configure AI large model]**.

3. Document Configuration

The car system is already configured with intent mapping files; customers experiencing the default gameplay can skip this step. To add new gameplay, please refer to these steps.

3.1 Create a md document

Open the [Intent Mapping] folder in this lesson's folder and select the corresponding [Intent Mapping_xx.txt] file based on your robot configuration.

 If you don't have software that can open Markdown documents, you can change the file format to .txt, view and modify the file content, and then change it back to .md format.

The screenshot shows a Typora window with the following content:

- Storage Location**
- Intent Mapping**
 - Command 1: "I'm at location 1 and I'm a little thirsty."
 - Intent Analysis 1: Find an area on the map with a bottle of mineral water, then go to that area to get a bottle.
 - Example: 1. Navigate to location A. 2. Check if a bottle of mineral water is present. 3. If a bottle of mineral water is present, report the distance to the user. Otherwise, move forward 0.5 meters and back up 0.5 meters. 4. Navigate to location 1 where the user is.
 - Command 2: "I want to sleep."
 - Intent Analysis 2: Tell the user a bedtime story to soothe them.
 - Command 3: "I'm a little hungry. Get me some bread."
 - Intent Analysis 2: Record the starting location, find an area on the map with bread, go to that area to observe the bread, report that the bread was found, and return to the starting location.
 - Example: 1. Navigate to location A. 2. Check if a bottle of mineral water is present. 3. If a bottle of bread is present, the servo nods. Otherwise, the servo shakes. Navigate to the starting location.
- Storage Location**
 - Master Bedroom
 - Toilet paper, pens, cell phone
 - Kitchen
 - Mineral water, cola, Sprite, bread

This example uses the [Intent Mapping] and [Storage Location] settings as examples (you can add multiple files and customize them).

[Intent Mapping]: Stores your intentions and desired large model robot control operations.

[Storage Location]: Stores your intentions and desired large model robot control operations.

3.2 Upload to the DIFY backend

1. Enter the robot's IP address:80 in your browser.
2. Click Add Knowledge, then click Create. Create a new knowledge base in Knowledge.

The screenshot shows the Dify Knowledge interface. At the top, there are tabs for Explore, Studio, Knowledge (selected), and Tools. Below the tabs, there's a search bar and a link to External Knowledge API. On the left, there's a sidebar with 'KNOWLEDGE' and 'API ACCESS' sections. A yellow arrow points from the 'Create Knowledge' button in the sidebar to the 'Intent Mapping_usb..md' document preview. Another yellow arrow points from the 'Intent Mapping_usb..md' preview to the 'Knowledge' tab at the top.

Directly drag the Intent Mapping_usb.md document provided in the tutorial into the repository.

The screenshot shows the 'DATA SOURCE' step of the Knowledge creation wizard. It has three tabs: STEP 1 DATA SOURCE, DOCUMENT PROCESSING, and EXECUTE & FINISH. Under 'Data Source', there are three options: Import from file (selected), Sync from Notion, and Sync from website. Under 'Upload file', there's a section to 'Drag and drop file or folder, or Browse'. A yellow arrow points from the 'Import from file' button in the 'Data Source' section to the 'Drag and drop file or folder, or Browse' section. Another yellow arrow points from the 'Intent Mapping_usb.md' file preview in the 'Upload file' section to the 'Next →' button.

After successfully adding the document, click Next.

The screenshot shows the 'DOCUMENT PROCESSING' step of the Knowledge creation wizard. It has three tabs: STEP 1 DATA SOURCE, DOCUMENT PROCESSING (selected), and EXECUTE & FINISH. Under 'Data Source', there are three options: Import from file (selected), Sync from Notion, and Sync from website. Under 'Upload file', there's a section to 'Drag and drop file or folder, or Browse'. A yellow arrow points from the 'Import from file' button in the 'Data Source' section to the 'Drag and drop file or folder, or Browse' section. A second yellow arrow points from the 'Intent Mapping_usb.md' file preview in the 'Upload file' section to the 'Next →' button.

Then enter the model's automatic slicing section, drag it to the back, and click Save & Process.

Index Method

- High Quality (RECOMMEND)**: Calling the embedding model to process documents for more precise retrieval helps LLM generate high-quality answers.
- Economical**: Using 10 keywords per chunk for retrieval, no tokens are consumed at the expense of reduced retrieval accuracy.

Embedding Model: text-embedding-v1

Retrieval Setting: Learn more about retrieval method, you can change this at any time in the Knowledge settings.

- Vector Search**: Generate query embeddings and search for the text chunk most similar to its vector representation.
- Rerank Model**: gte-rerank
- Top K**: 3
- Score Threshold**: 0.5
- Full-Text Search**: Index all terms in the document, allowing users to search any term and retrieve relevant text chunk containing those terms.
- Hybrid Search (RECOMMEND)**: Execute full-text search and vector searches simultaneously, re-rank to select the best match for the user's query. Users can choose to set weights or configure to a Rerank model.

Save & Process

PREVIEW: Intent Mapping usb.md | 0 ESTIMATED CHUNKS

Click the 'Preview Chunk' button on the left to load the preview

Next, click Go to document

Knowledge created: We automatically named the Knowledge, you can modify it at any time.

Knowledge name: Intent Mapping_usb...

Embedding completed: Intent Mapping_usb.md

Chunking Setting: Custom

Maximum Chunk Length: 1024

Text Preprocessing Rules: Replace consecutive spaces, newlines and tabs

Index Method: High Quality

Retrieval Setting: Vector Search

Access the API

Go to document

What's next: After the document finishes indexing, the Knowledge can be integrated into the application as context, you can find the context setting in the prompt orchestration page. You can also use it as an independent ChatGPT indexing plugin for reference.

You will now see the new knowledge base.

Documents: All files of the Knowledge are shown here, and the entire Knowledge can be linked to Dify citations or indexed via the Chat plugin. Learn more?

| NAME | CHUNKING MODE | WORDS | RETRIEVAL COUNT | UPLOAD TIME | STATUS | ACTION |
|-----------------------|---------------|-------|-----------------|---------------------|-----------|-------------|
| Intent Mapping_usb.md | GENERAL | 1.2k | 0 | 09/19/2025 07:46 AM | Available | Edit |

Documents: Intent Mapping_usb... LOCAL DOCS

NO LINKED APPS: Retrieval Testing, Settings

1 / 1

Click Studio and select the decision layer for the corresponding version.

The screenshot shows the Dify Studio interface. At the top, there are navigation tabs: Explore, Studio (which is selected and highlighted with a yellow border), Knowledge, and Tools. Below the tabs, there are search and filter options: All, Workflow, Chatflow, Chatbot, Agent, Completion, Created by me, All Tags, and Search. A sidebar on the left titled 'CREATE APP' offers options: Create from Blank, Create from Template, and Import DSL file. The main area displays three AI applications:

- Execution layer AI-NUWA**: Edited at 09/06/2025 6:28. Description: Generate an action list and responses to users.
- Execution layer AI**: Edited at 09/06/2025 6:21. Description: Generate an action list and responses to users.
- Decision-making level AI**: Edited at 09/06/2025 11:25. Description: Decision-making level AI.

At the bottom left, there's a 'Join the community' section with a link to 'Discuss with team members, contributors and developers on different channels.' On the right, there's a placeholder for dropping a DSL file: 'Drop DSL file here to create app'.

After entering, select Add Knowledge Base and add the intent mapping you just created.

The screenshot shows the Dify Studio interface for the 'Decision-making level AI' application. The top navigation bar includes 'Explore', 'Studio' (selected), 'Decision-making lev...', 'Knowledge', 'Tools', and 'Plugins'. The left sidebar shows 'Orchestrate' (selected) and other options: API Access, Logs & Annotations, and Monitoring. The main area has a large blue box labeled 'INSTRUCTIONS' containing the 'Role' and 'Work Flow' sections. To the right is the 'Debug & Preview' section, which includes a 'Talk to Bot' interface with a message 'Features Enabled' and a 'Manage' button. In the 'Retrieval Setting' section, there is a 'Variables' list with 'Training sample' and a 'METADATA FILTERING' section with 'Vision'. A yellow arrow points to the '+ Add' button next to the 'Retrieval Setting' dropdown.

The screenshot shows the Orchestrate AI interface. On the left, there's a sidebar with a 'Orchestrate' tab. The main area has a blue header bar with 'INSTRUCTIONS' and a 'Generate' button. Below it is a large text box containing two sections: 'Role' and 'Work Flow'. The 'Role' section describes the AI agent as a decision-making layer AI agent that controls a physical robot. The 'Work Flow' section instructs the user to analyze the instruction execution program and use combinations of available actions from the robot action function library to form task steps. A small number '4287' is at the bottom left of this box. Below this is a 'Variables' section with a '+ Add' button and a note about variables. Under 'Knowledge', there are two items: 'Intent Mapping_usb...' (HQ-VECTOR) and 'Training sample' (ECO-INVERTED). A yellow box highlights these two items. Below 'Knowledge' is a 'METADATA FILTERING' section with a 'Vision' item, a 'Settings' button, and a 'Disabled' dropdown. On the right side of the interface, there's a 'Debug' tab.

In the text dialog on the right, you can test the effectiveness of your customized intent understanding. If it doesn't meet your expectations, try adjusting the semantics in the intent plan until it meets your expectations.

4. Run the Example

4.1 Starting the Program

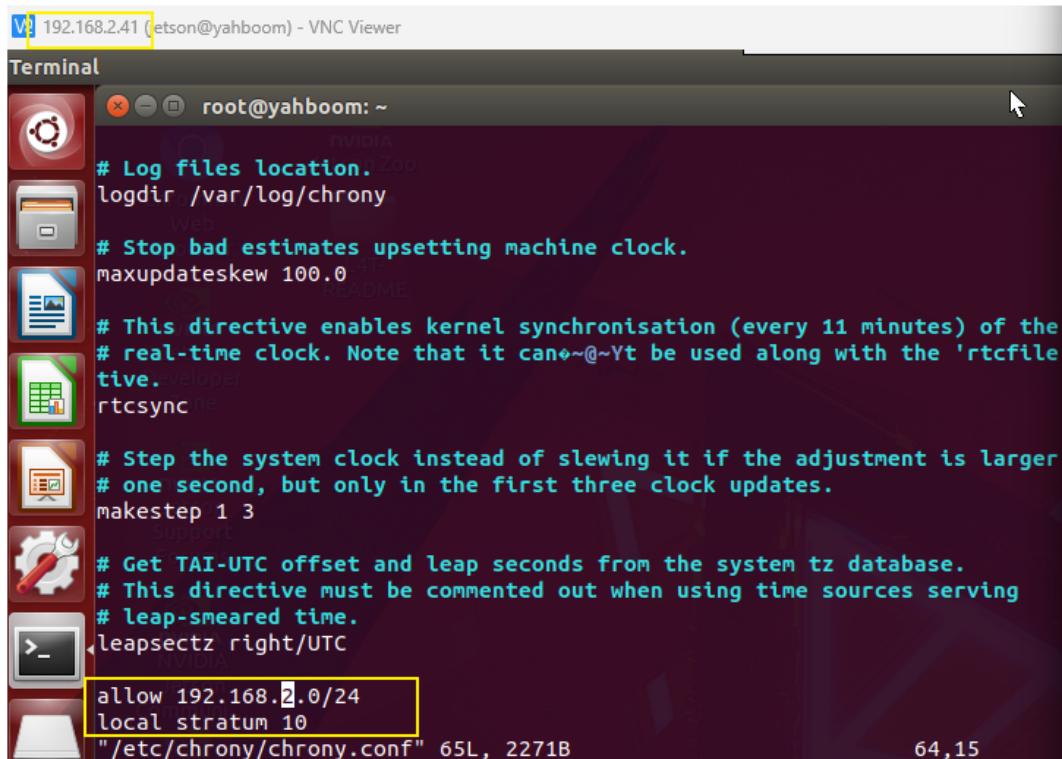
4.1.1 Jetson Nano Board Startup Steps:

Due to Nano performance issues, navigation-related nodes must be run on a virtual machine. Therefore, navigation performance is highly dependent on the network. We recommend running in an indoor environment with a good network connection. You must first configure the following:

- **Board Side (Requires Docker)**

Open a terminal and enter

```
sudo vi /etc/chrony/chrony.conf
```



```
V 192.168.2.41 (jetson@yahboom) - VNC Viewer
Terminal
root@yahboom: ~
# Log files location.
logdir /var/log/chrony
# Stop bad estimates upsetting machine clock.
maxupdateskew 100.0
# This directive enables kernel synchronisation (every 11 minutes) of the
# real-time clock. Note that it can't be used along with the 'rtcsync' directive.
rtcsync
# Step the system clock instead of slewing it if the adjustment is larger
# one second, but only in the first three clock updates.
makestep 1 3
# Get TAI-UTC offset and leap seconds from the system tz database.
# This directive must be commented out when using time sources serving
# leap-smeared time.
leapsectz right/UTC
allow 192.168.2.0/24
local stratum 10
"/etc/chrony/chrony.conf" 65L, 2271B
```

Add the following two lines to the end of the file: (Fill in 192.168.x.0/24 based on the actual network segment; this example uses the current board IP address of 192.168.2.41.)

```
allow 192.168.x.0/24 #x indicates the corresponding network segment
local stratum 10
```

After saving and exiting, enter the following command to take effect:

```
sudo service chrony restart
```

```
root@yahboom:~# sudo service chrony restart
 * Stopping time daemon chronyd
 * Starting time daemon chronyd
[ OK ]
[ OK ]
root@yahboom:~#
```

- **Virtual Machine**

Open a terminal and enter

```
echo "server 192.168.2.41 iburst" | sudo tee
/etc/chrony/sources.d/jetson.sources
```

The 192.168.2.41 entry above is the board's IP address.

Enter the following command to take effect.

```
sudo chronyc reload sources
```

Enter the following command again to check the latency. If the IP address appears, it's normal.

```
chronyc sources -v
```

```
yahboom@VM:~$ chronyc sources -v
.-- Source mode '^' = server, '=' = peer, '#' = local clock.
/ .- Source state '*' = current best, '+' = combined, '-' = not combined,
| /           'x' = may be in error, '~' = too variable, '?' = unusable.
||                               .- xxxx [ yyyy ] +/- zzzz
||   Reachability register (octal) -. | xxxx = adjusted offset,
||   Log2(Polling interval) --. | yyyy = measured offset,
||                                \ | zzzz = estimated error.
||                                | |
MS Name/IP address      Stratum Poll Reach LastRx Last sample
=====
^- prod-ntp-5.ntp1.ps5.cano>    2    7   277   133  -4048us[-5395us] +/- 132ms
^- alphyn.canonical.com        2    8   177    66  -193us[-193us] +/- 134ms
^- prod-ntp-3.ntp1.ps5.cano>    2    8   367   131   -16ms[-18ms] +/- 127ms
^- prod-ntp-4.ntp4.ps5.cano>    2    8   377   129  -4907us[-6254us] +/- 133ms
** time.nju.edu.cn            1    7   377    69   +77us[-1270us] +/- 17ms
^+ 139.199.215.251           2    8   377   135  +2358us[+1011us] +/- 46ms
^+ 119.28.183.184             2    7   377   193  +2061us[+713us] +/- 28ms
^+ 119.28.206.193             2    8   367     8  +2058us[+2058us] +/- 36ms
^+ 192.168.2.41               4    6   377     6   -12ms[-12ms] +/- 92ms
yahboom@VM:~$
```

- Start the program

On the mainboard, open a terminal in Docker and enter the command:

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

```
File Edit Tabs Help
root@raspb... x root@raspb... x
root@raspb... ~# ros2 launch largemodel largemodel_control.launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2025-08-21-16-01-10-206398-raspberrypi-98016
[INFO] [launch]: Default logging verbosity is set to INFO

----- robot_type = A1, rplidar_type = tmini, camera_type = nova -----
[INFO] [robot_type-1]: process started with pid [98002]
[INFO] [rplidar_ros2_driver_node-10]: process started with pid [98050]
[INFO] [ascamera_node-1]: process started with pid [98050]
[INFO] [joint_state_publisher-2]: process started with pid [98052]
[INFO] [robot_state_publisher-3]: process started with pid [98054]
[INFO] [Ackman_driver_A1-4]: process started with pid [98056]
[INFO] [base_node_A1-5]: process started with pid [98058]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [98061]
[INFO] [ekf_node-7]: process started with pid [98065]
[INFO] [yahboom_joy_A1-8]: process started with pid [98075]
[INFO] [joy_node-9]: process started with pid [98078]
[INFO] [static_transform_publisher-11]: process started with pid [98090]
[INFO] [static_transform_publisher-12]: process started with pid [98092]
[INFO] [static_transform_publisher-13]: process started with pid [98095]
[INFO] [model_service-14]: process started with pid [98100]
[INFO] [action_service_nuwa-15]: process started with pid [98102]
[INFO] [asr-16]: process started with pid [98105]
[robot_state_publisher-3] [INFO] [1755763271.586916300] [robot_state_publisher]: got segment base_footprint
[robot_state_publisher-3] [INFO] [1755763271.587125040] [robot_state_publisher]: got segment base_link
[robot_state_publisher-3] [INFO] [1755763271.587149958] [robot_state_publisher]: got segment camera_link
[robot_state_publisher-3] [INFO] [1755763271.587160632] [robot_state_publisher]: got segment imu_link
[robot_state_publisher-3] [INFO] [1755763271.587171336] [robot_state_publisher]: got segment laser
[robot_state_publisher-3] [INFO] [1755763271.587180410] [robot_state_publisher]: got segment left_front_wheel_joint
[robot_state_publisher-3] [INFO] [1755763271.587189632] [robot_state_publisher]: got segment left_rear_wheel_hinge
[robot_state_publisher-3] [INFO] [1755763271.587199428] [robot_state_publisher]: got segment left_steering_hinge_joint
[robot_state_publisher-3] [INFO] [1755763271.587208447] [robot_state_publisher]: got segment right_front_wheel_joint
[robot_state_publisher-3] [INFO] [1755763271.587217132] [robot_state_publisher]: got segment right_rear_wheel_hinge
[robot_state_publisher-3] [INFO] [1755763271.587226428] [robot_state_publisher]: got segment right_steering_hinge_joint
[robot_state_publisher-3] [INFO] [1755763271.587232734] [robot_state_publisher]: got segment right_steer_axle_hinge
```

On the virtual machine, create a new terminal and start.

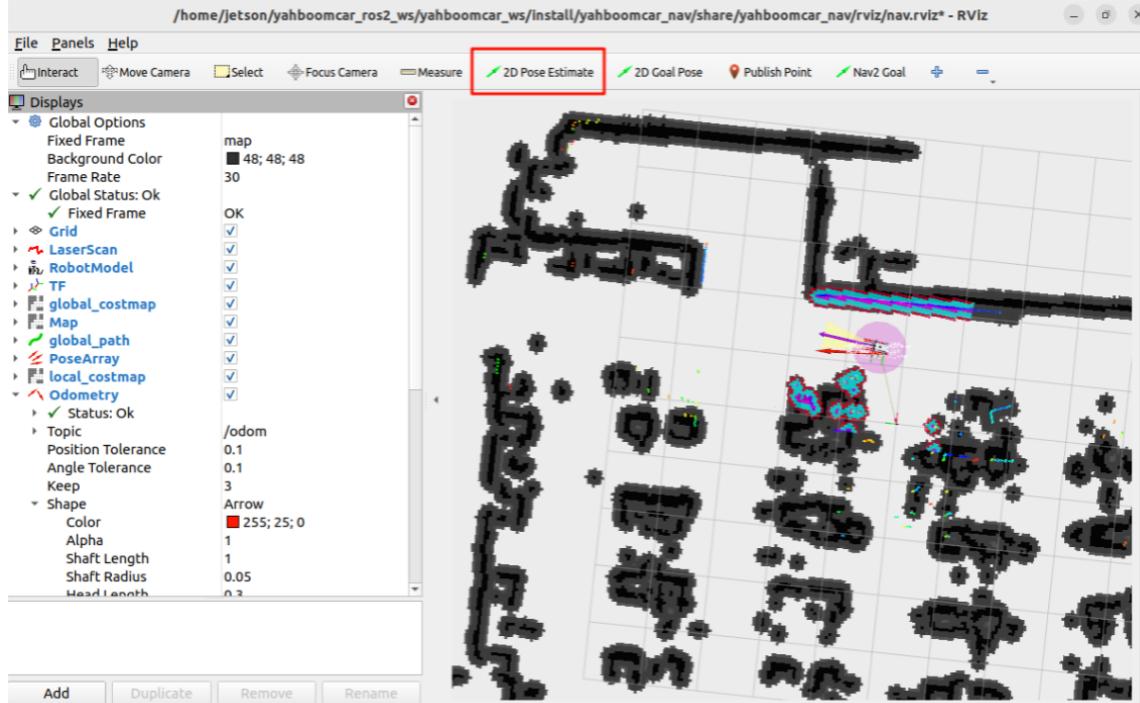
```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start before displaying the map. Then open a VM terminal and enter:

```
ros2 launch yahboomcar_nav navigation_teb_launch.py
```

Note: When running the car's mapping function, you must enter the save map command on the VM to save the navigation map.

After that, follow the navigation function startup process to initialize positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to select it. Roughly mark the robot's position and orientation on the map. After initializing positioning, preparations are complete.



4.1.2 Other main control startup steps:

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

Open a terminal in Docker and enter the following command:

```
ros2 launch largemode largemode_control.launch.py text_chat_mode:=True
```

```
root@raspberrypi:~
```

The terminal window shows the command being run and its output. The output includes several INFO messages from different nodes starting up, such as ydlidar_ros2_driver_node-10, ascamera_node-1, joint_state_publisher-2, robot_state_publisher-3, Ackman_driver_A1-4, base_node_A1-5, imu_filter_madgwick_node-6, ekf_node-7, yahboom_joy_A1-8, joy_node-9, static_transform_publisher-11, static_transform_publisher-12, static_transform_publisher-13, model_service-14, action_service_nuwa-15, and ascamera_node-1. The log also shows camera parameters like depth_width, depth_height, rgb_width, and rgb_height being set, along with other initialization details.

Create a new terminal on the virtual machine and start the command.(**For Raspberry Pi and Jetson Nano, it is recommended to run the visualization in the virtual machine**)

```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start and the image will be displayed. Then open the Docker terminal and enter

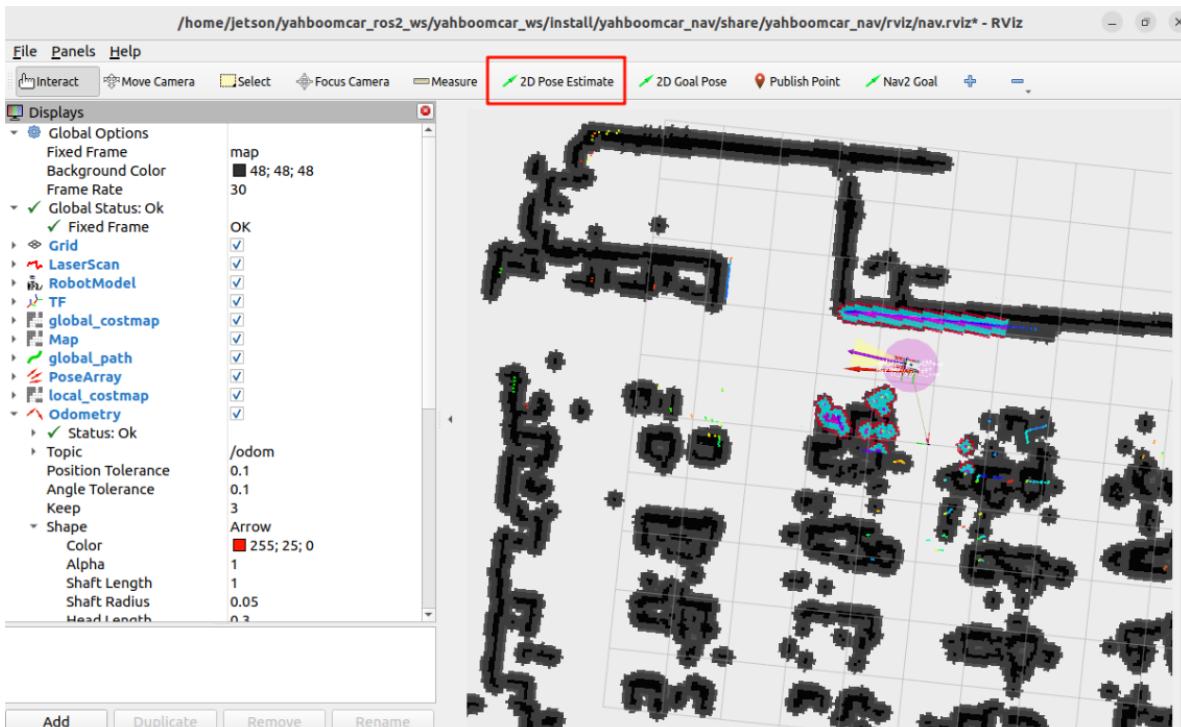
```
# Choose one of the two navigation algorithms
# Standard navigation
ros2 launch yahboomcar_nav navigation_teb_launch.py

#Fast relocalization navigation (RDKX5 and Raspberry Pi 5 are not supported)
ros2 launch yahboomcar_nav localization_imu_odom.launch.py use_rviz:=true
load_state_filename:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream

ros2 launch yahboomcar_nav navigation_cartodwb_launch.py
maps:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.yaml
params_file:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/params/cartoteb_nav_params.yaml
```

Note: yahboomcar.yaml and yahboomcar.pbstream must be mapped simultaneously, meaning they are the same map. Refer to the cartograph mapping algorithm for saving maps.

After that, follow the navigation function startup process to initialize positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to enter the selection state. Roughly mark the robot's position and orientation on the map. After initializing positioning, preparations are complete.



Log into the same docker run from multiple terminals and start it.

```
ros2 run text_chat text_chat
```

```
File Edit Tabs Help
* root@raspb... * root@raspb... * root@raspb... * root@raspb...
root@raspberrypi:/# ros2 run text_chat text_chat
user input: [ ]
```

4.2 Test Cases

Here are some test cases for reference; users can create their own dialogue commands.

- I'm in the office and feeling thirsty.

4.2.1 Example 1

Type "I'm in the office now, feeling a little thirsty" in the terminal. The terminal prints the following information:

```
File Edit Tabs Help
root@raspb... * root@raspb... * root@raspb... * pi@raspberr...
root@raspberrypi:~# ros2 run text_chat text_chat
user input: 我现在办公区，我有点口渴了
okay@, let me think for a moment... -[INFO] [1755692529.223522879] [text_chat_node]: 决策层AI规划:1. 导航前往厨房区域
2. 观察是否存在矿泉水
3. 如果观察到矿泉水，反馈矿泉水和你之间的距离；否则前进0.5米后退0.5米
4. 导航返回办公区
user input: [INFO] [1755692530.957639015] [text_chat_node]: "action": ["navigation(G)", "response": "好的，我这就去厨房给你找矿泉水，就像一个贴心的小助手一样"]
[INFO] [1755692557.989323529] [text_chat_node]: "action": ["seewhat()"], "response": "我已经到达厨房了，现在用我的火眼金睛寻找矿泉水"
[INFO] [1755692566.396245271] [text_chat_node]: "action": ["get_dist(240,210)"], "response": "我看到前面有一个矿泉水瓶，正在测量它和我的距离"
[INFO] [1755692564.370085347] [text_chat_node]: "action": ["navigation(F)", "response": "我已经测到矿泉水离我1.004米，现在准备返回办公区啦"]
[INFO] [1755692590.463255325] [text_chat_node]: "action": ["finishtask()"], "response": "我已经完成全部任务啦，矿泉水在1.004米处，现在安全返回办公区了"
[ ]
```

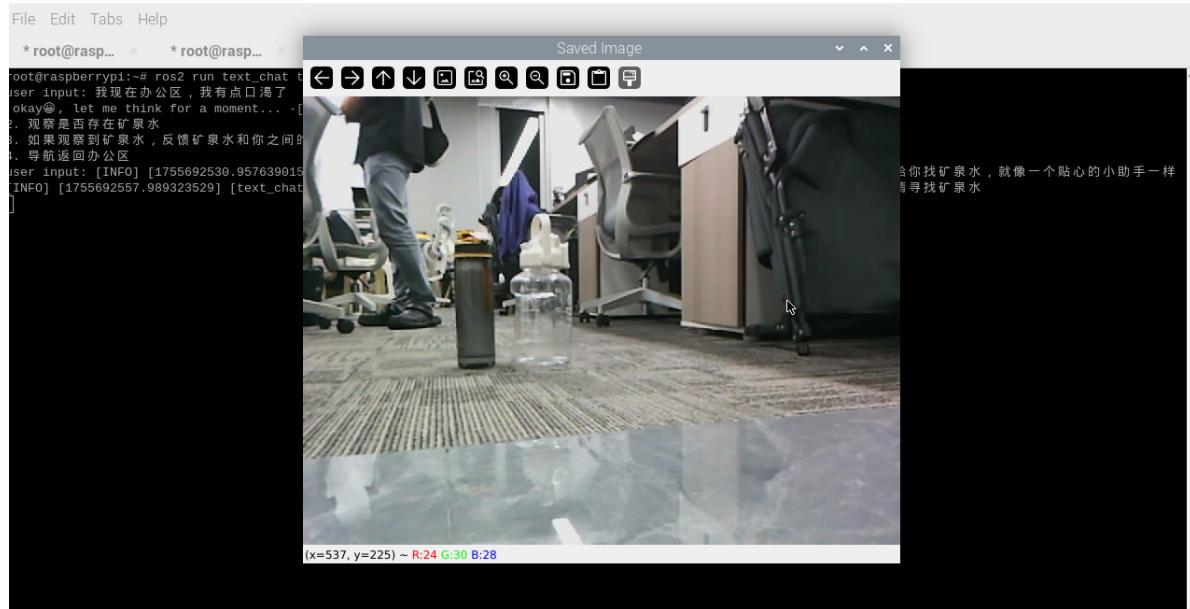
The decision-making model outputs the planned task steps:

```
user input: [ ]
```

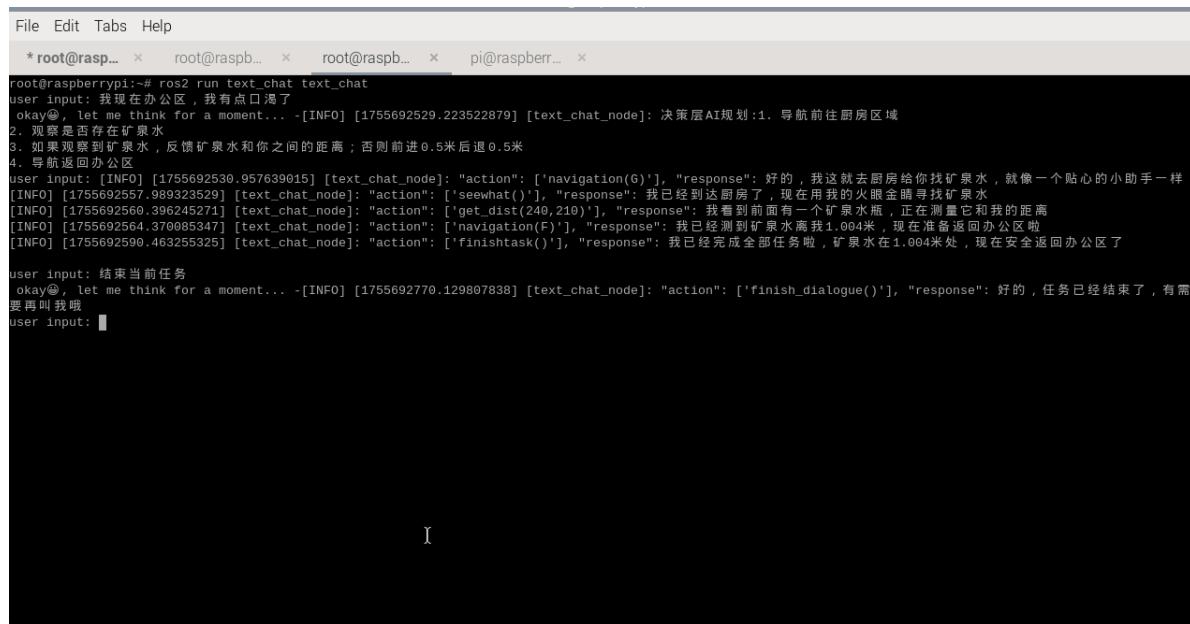
The execution model then executes the task steps:

```
user input: [INFO] [1755692530.957639015] [text_chat_node]: "action": ["navigation(G)", "response": "好的，我这就去厨房给你找矿泉水，就像一个贴心的小助手一样"]
[INFO] [1755692557.989323529] [text_chat_node]: "action": ["seewhat()"], "response": "我已经到达厨房了，现在用我的火眼金睛寻找矿泉水"
[INFO] [1755692566.396245271] [text_chat_node]: "action": ["get_dist(240,210)"], "response": "我看到前面有一个矿泉水瓶，正在测量它和我的距离"
[INFO] [1755692564.370085347] [text_chat_node]: "action": ["navigation(F)", "response": "我已经测到矿泉水离我1.004米，现在准备返回办公区啦"]
[INFO] [1755692590.463255325] [text_chat_node]: "action": ["finishtask()"], "response": "我已经完成全部任务啦，矿泉水在1.004米处，现在安全返回办公区了"
[ ]
```

A four-second screen appears after arriving at the kitchen.



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 press the **ENTER** key in the terminal and continue the conversation by entering the "End current task" command to terminate the current task cycle and start a new one.



5. Source Code Analysis

Source code is located at:

Jetson Orin Nano host:

```
#NUWA Camera User
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_nuwa.py
#USB Camera User
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_usb.py
```

RDK X5:

```
#NUWA Camera User  
/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_  
service_nuwa.py  
#USB Camera User  
/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_  
service_usb.py
```

Jetson Nano, Raspberry Pi Host:

You need to first enter Docker

```
#NUWA Camera User  
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_  
nuwa.py  
#USB Camera User  
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemode1/largemode1/action_service_  
usb.py
```

5.1 Example 1

action_service.py :

This example uses the **seewhat**, **navigation**, **load_target_points**, and **get_dist** methods in the **CustomActionServer** class. **seewhat**, **navigation**, **load_target_points**, and **get_dist** are described in the previous section [2.Multimodal visual understand], [6.Multimodal visual understand + Depth Camera Distance Question Answering], [7.Multimodal visual understand+SLAM navigation] have all been explained, and there are no new procedures in this chapter.