Multimodal visual understanding + SLAM navigation

1. Course Content

1. Learn to use the robot's visual understanding and SLAM navigation functions
2. Study the new key source code in the tutorial

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

Note: This chapter requires you to complete at least one SLAM map according to the LiDAR chapter course.

2.3 Configuring map mapping files

For more details on the principles and concepts of map mapping, please refer to the course content of [2. Basic Knowledge of AI Large Model - 3. Embodied Intelligent Robot System Architecture]

Connect to the robot desktop via VNC, create two new terminals, and start the first terminal first.

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

Then start in the second terminal

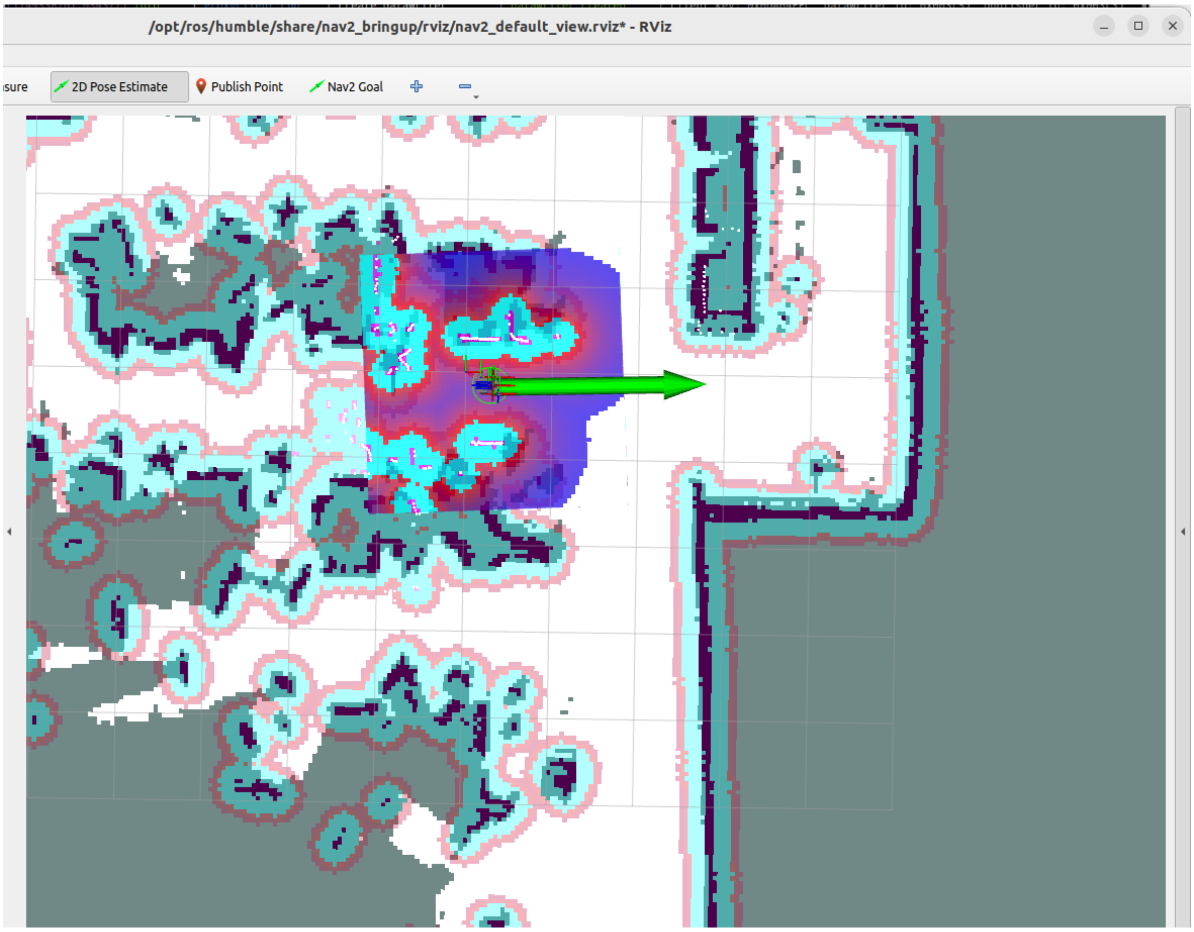
```
ros2 launch M3Pro_navigation navigation2.launch.py
```

Create a new terminal on the virtual terminal and start

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

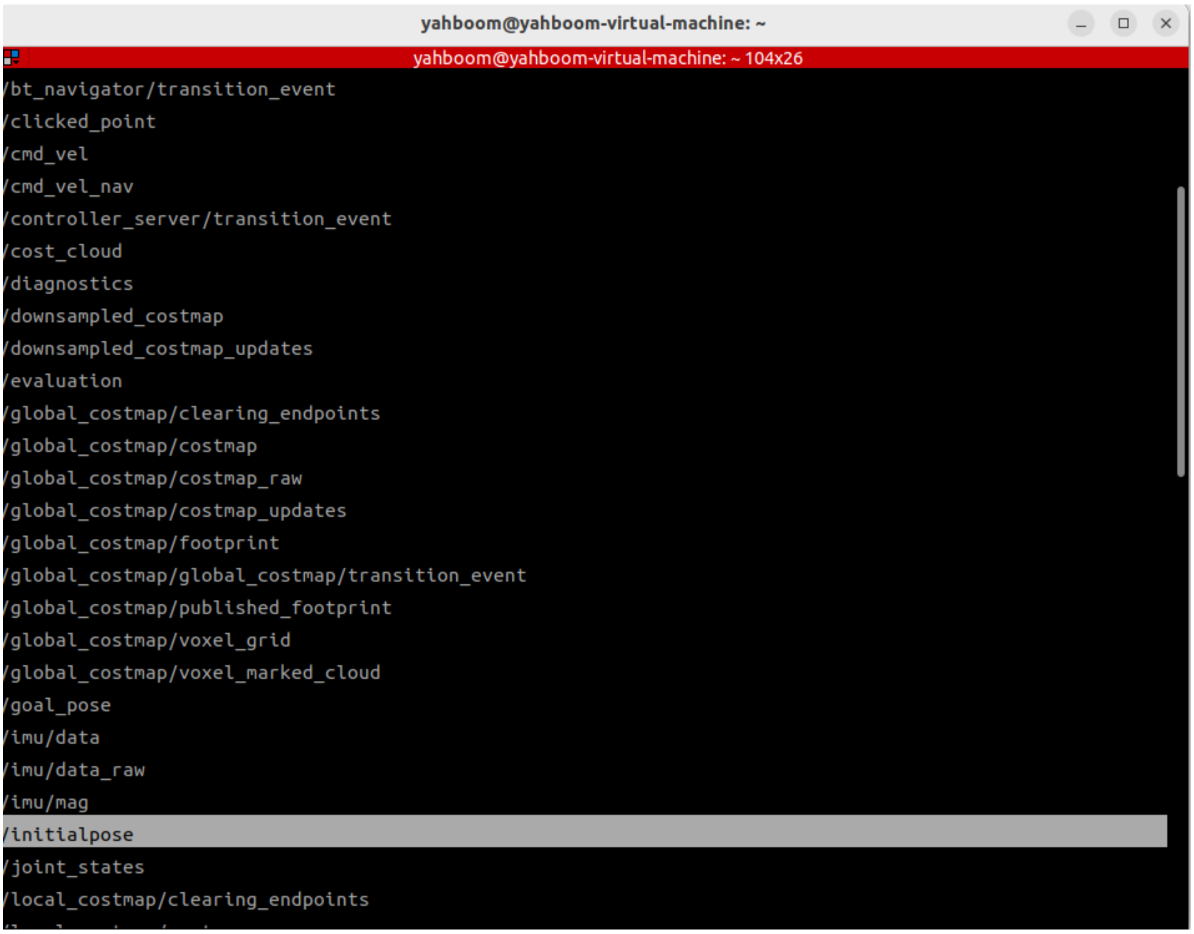
The rviz2 visualization interface will then open. Click **2D Pose Estimate** in the upper toolbar to enter the selection state and roughly mark the robot's position and orientation on the map.

The robot's model will be displayed on the map as shown below:



Create a new terminal in the virtual machine again and enter the command

```
ros2 topic list
```

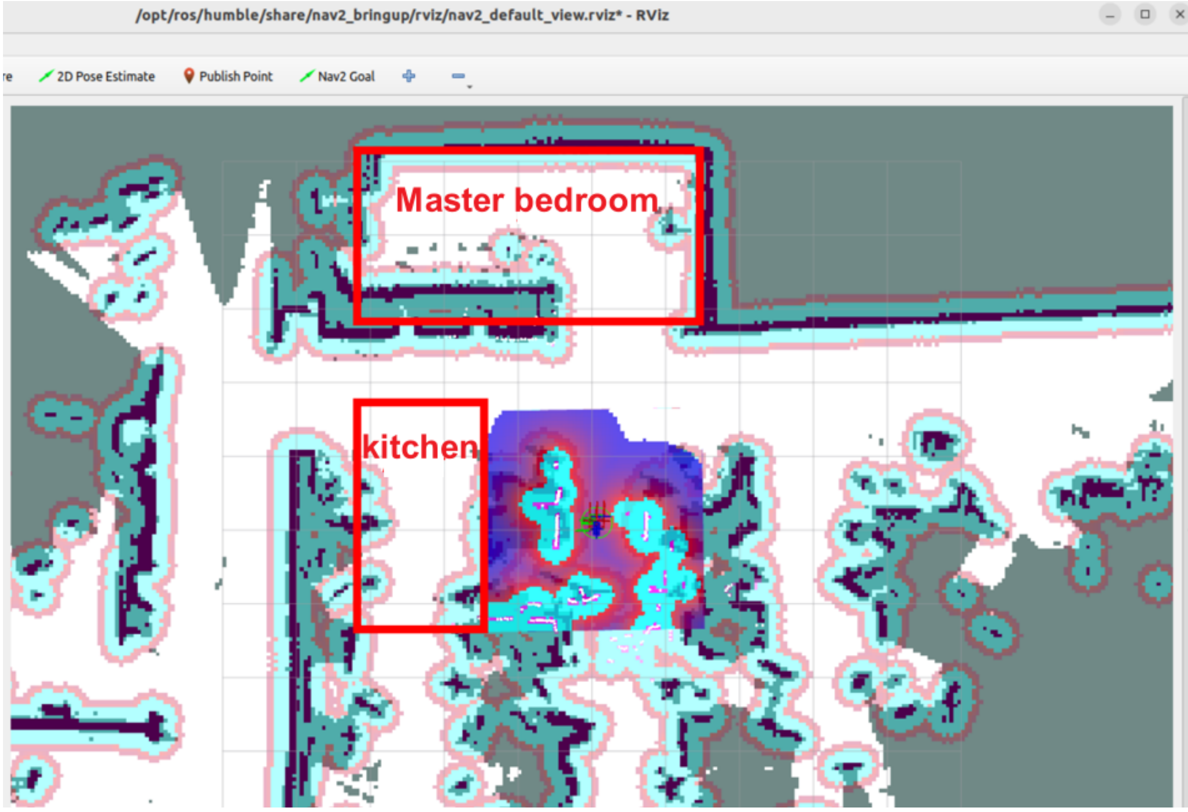


You can see the topic **/initialpose , which is the topic for using the 2D Pose Estimate** tool in rviz2 in the previous step . We can use this tool to publish coordinates to view the data of the **/initialpose** topic to obtain the coordinates and orientation angle of a certain point on the map.

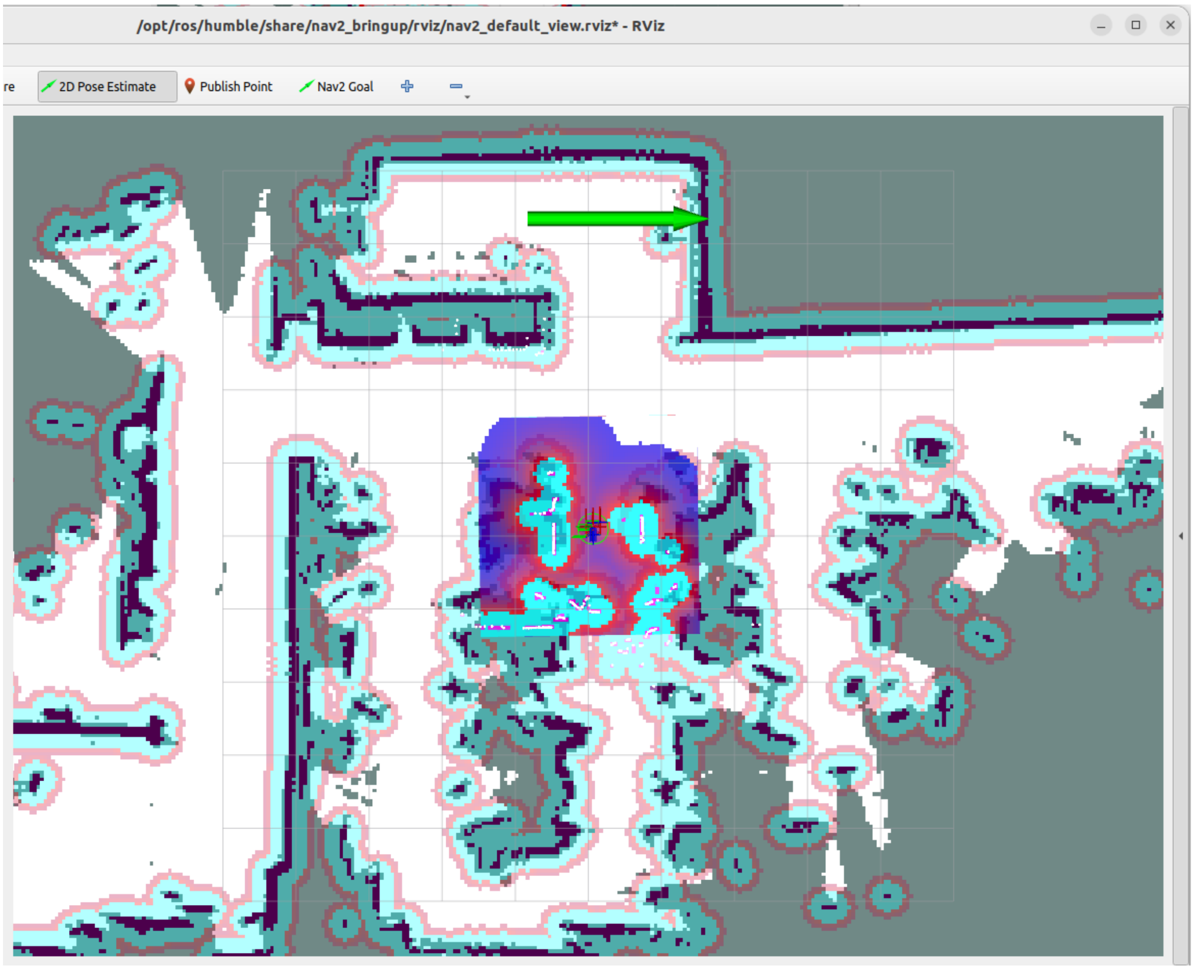
Enter the following command in the terminal to observe the data on the **/initialpose topic:**

```
ros2 topic echo /initialpose
```

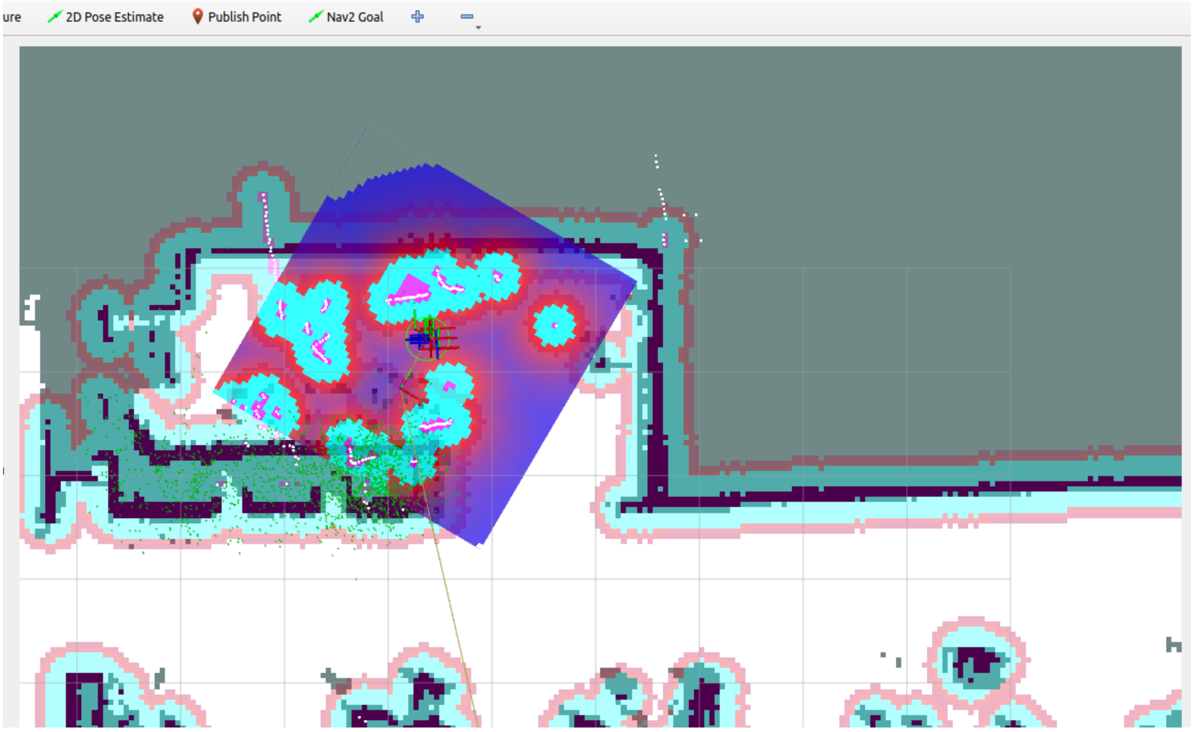
We can name any point on the map. Here we take "Master Bedroom" and "Kitchen" as examples.



As shown in the figure below, we first click the **2D Pose Estimate** tool, then click and hold the left button of the mouse anywhere in the "Master Bedroom" area to adjust the direction, and release it after confirming.



The robot position will be adjusted to the position and orientation just selected. You can preview the robot's expected position and orientation in rviz



In the last frame of the terminal window, you can see the coordinates of the target point we just marked with the **2D Pose Estimate tool.**

```
yahboom@yahboom-virtual-machine: ~
yahboom@yahboom-virtual-machine: ~ 104x26
yahboom@yahboom-virtual-machine:~$ ros2 topic echo /initialpose
header:
  stamp:
    sec: 1749544815
    nanosec: 989423507
  frame_id: map
pose:
  pose:
    position:
      x: 4.317397594451904
      y: 0.4287700653076172
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.7071081943275359
      w: 0.707105368042735
  covariance:
  - 0.25
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
```

Open `map_mapping.yaml` the map file, which is located at:

jetson orin nano, jetson orin NX host:

```
/home/jetson/M3Pro_ws/src/largemodel/config/map_mapping.yaml
```

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

```
root/M3Pro_ws/src/largemodel/config/map_mapping.yaml
```

Replace the "name" in A with "master bedroom", and the "position" and "orientation" data with the data we just saw in the terminal **/initialpose topic**

```
#根据实际的场景环境，自定义地图中的区域，可以添加任意个区域，注意和大模型的地图映射保持一致即可
#According to the actual scene environment, customize the areas in the map. You can add any number of areas, just make sure they are consistent with the map mapping of the large model
#地图映射Map mapping
A:
  name: 'python'
  position:
    x: 4.317397594451904
    y: 0.4287700653076172
  orientation:
    x: 0.0
    y: 0.0
    z: -0.7071081943275359
    w: 0.707105368042735

#此处可新增添加区域对应的栅格地图坐标点，注意和上面格式保持一致
#Here, you can add the raster map coordinate points corresponding to the added area. Please note that the format should be consistent with the above
```

In the same way, we can add a map of the kitchen

```
#根据实际的场景环境，自定义地图中的区域，可以添加任意个区域，注意和大模型的地图映射保持一致即可
#According to the actual scene environment, customize the areas in the map. You can add any number of areas, just make sure they are consistent with the map mapping of the large model
#地图映射Map mapping
A:
  name: 'Master bedroom'
  position:
    x: 4.317397594451904
    y: 0.4287700653076172
  orientation:
    x: 0.0
    y: 0.0
    z: -0.7071081943275359
    w: 0.707105368042735
B:
    name: 'kitchen'
    position:
      x: 1.1062586307525635
      y: 2.337576150894165
    orientation:
      x: 0.0
      y: 0.0
      z: 0.9999804570072698
      w: 0.006251848009342877
```

```
#此处可新增添加区域对应的栅格地图坐标点，注意和上面格式保持一致
#Here, you can add the raster map coordinate points corresponding to the added area. Please note that the format should be consistent with the above
```

Then switch to the workspace in the vehicle terminal `M3Pro_ws/`: recompile the largemodel function package to take effect

jetson orin nano, jetson orin NX host:

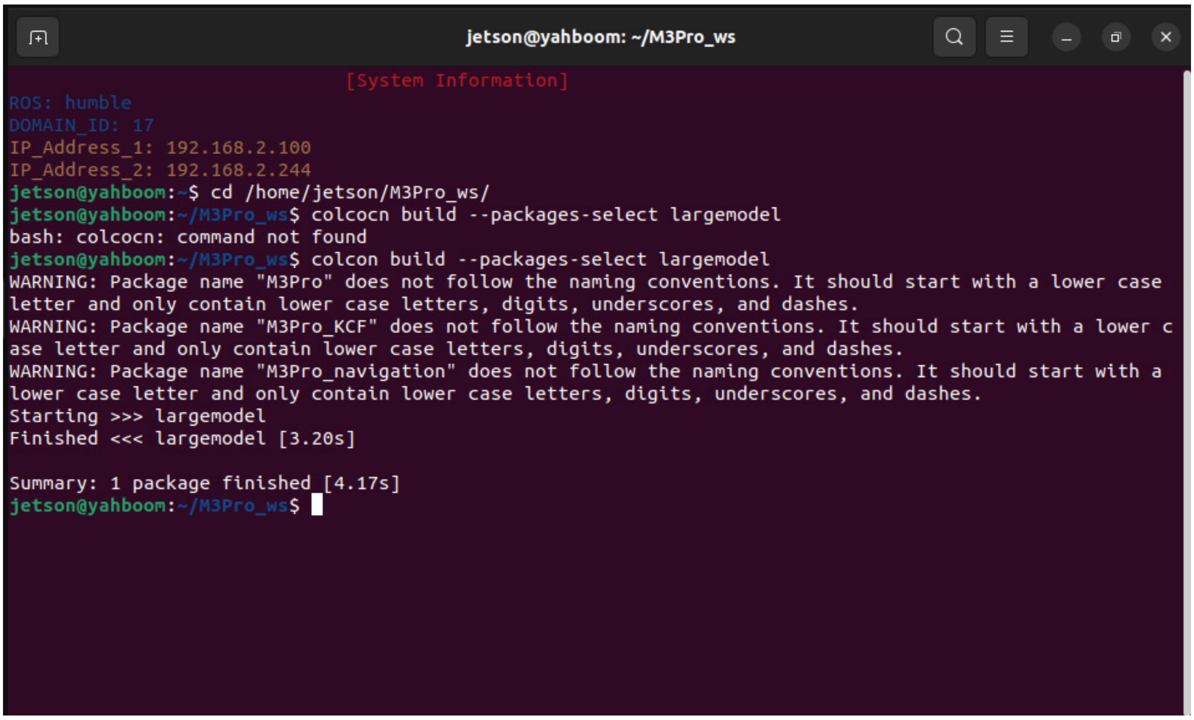
```
cd /home/jetson/M3Pro_ws/
```

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

```
cd root/M3Pro_ws/
```

Recompile the feature package:

```
colcon build --packages-select largemodel
```



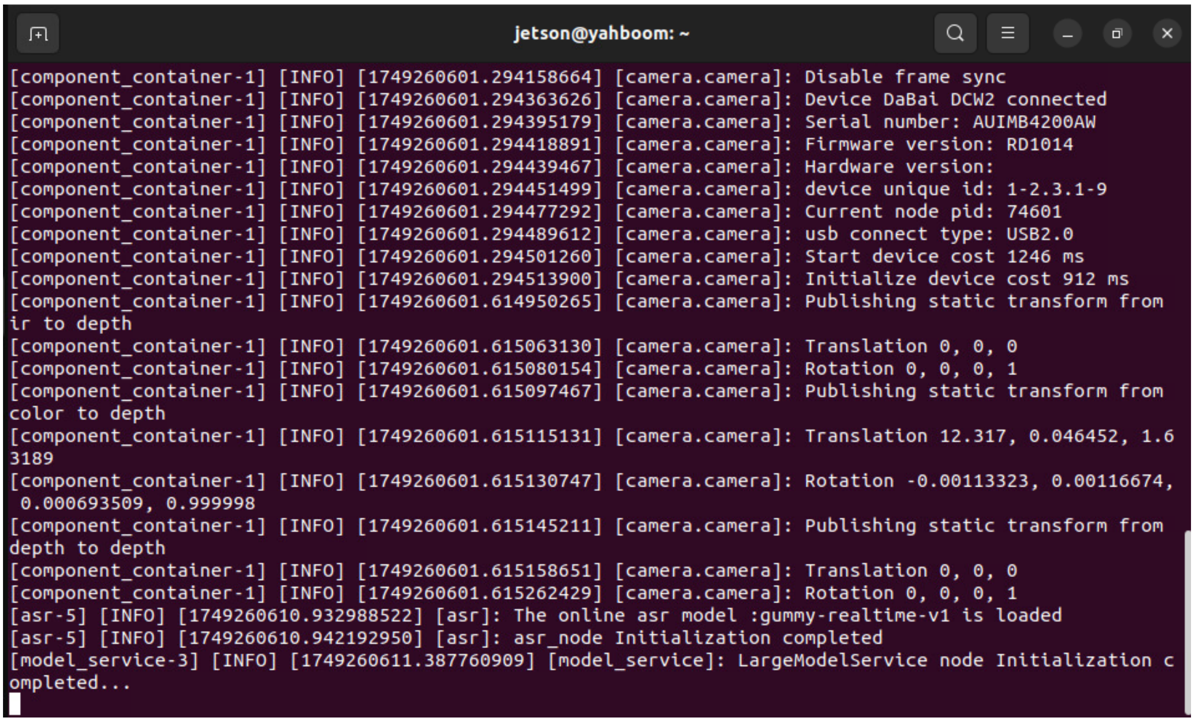
3. Run the case

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



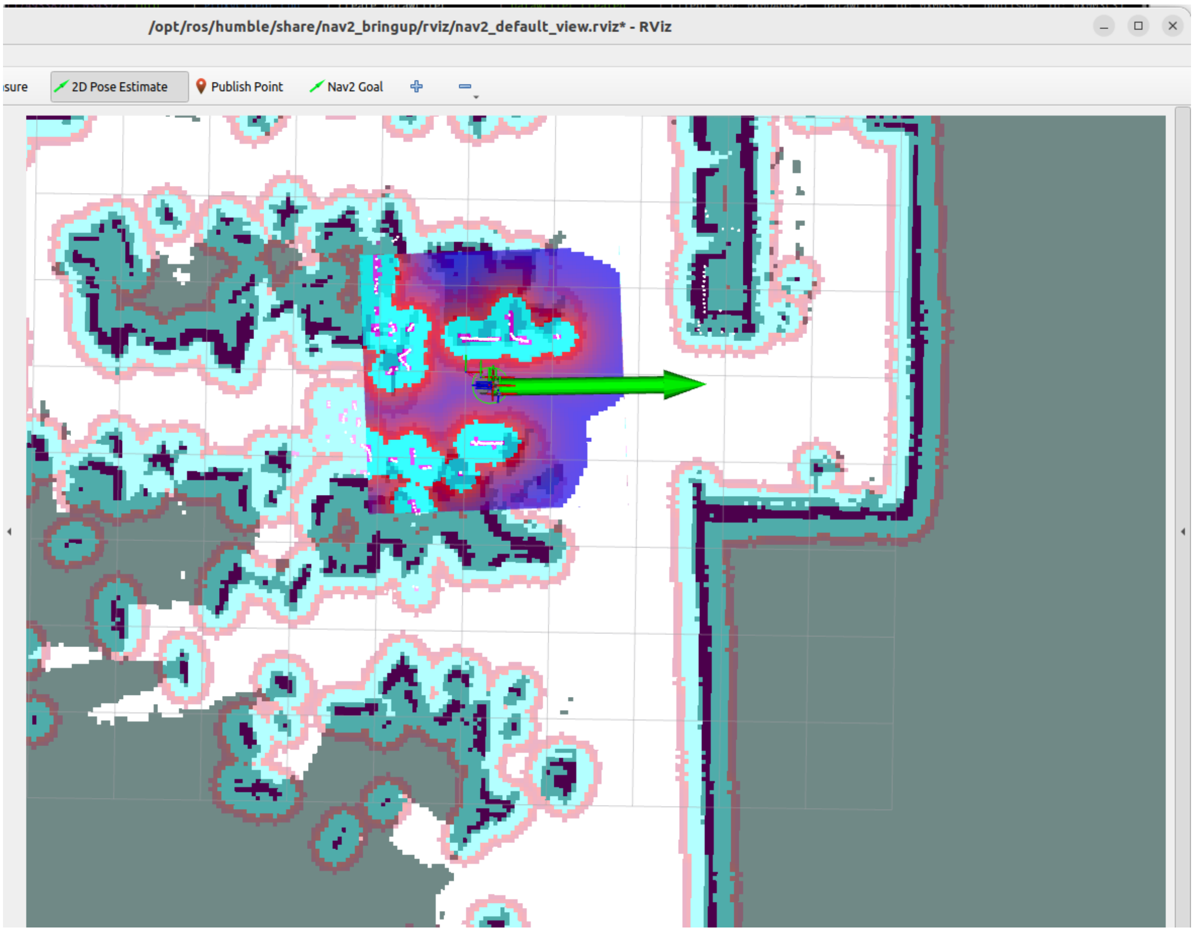
Open two terminals on the vehicle and enter the following commands:

```
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.

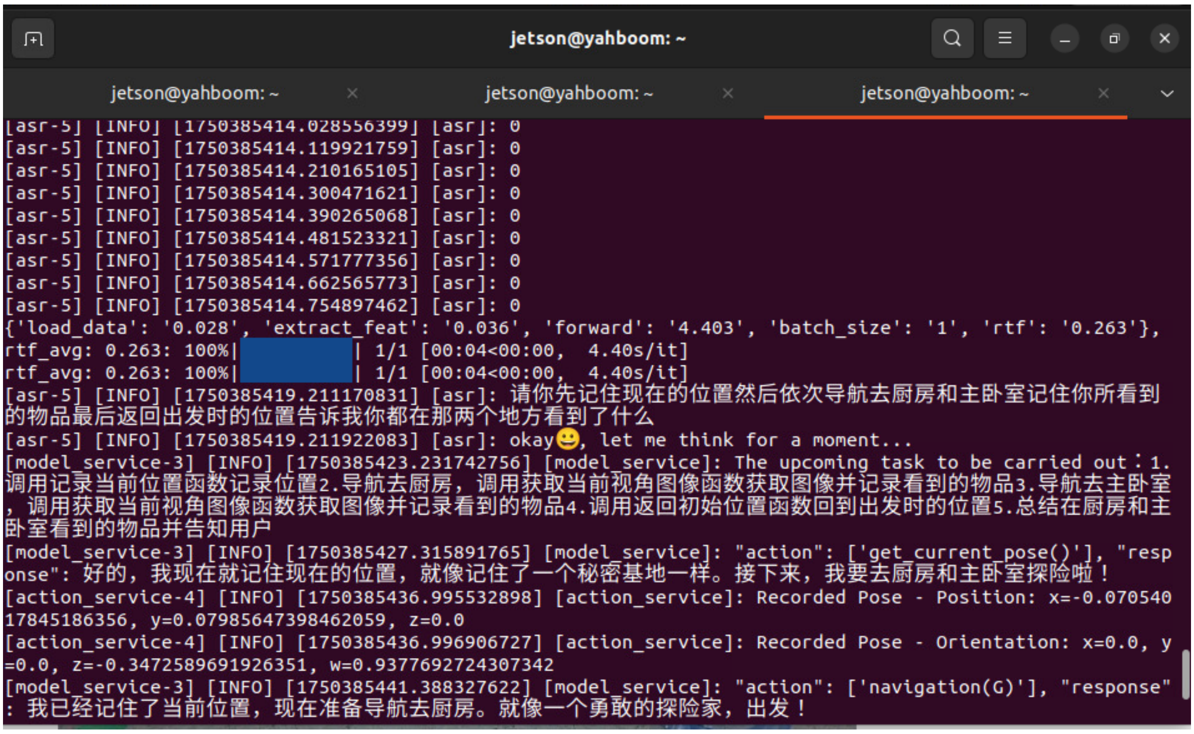
- Please remember your current location first, then navigate to the kitchen and master bedroom in turn and remember what you see. Finally, return to your starting location and tell me what you see in each of these two places.

3.2.1 Case 1

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:

The execution layer model will execute the subtasks according to the task steps planned by the decision layer model. 1. Navigate to the "master bedroom" and observe the objects in the environment

Place a pen in the simulated "Master Bedroom" and a pack of toilet paper in the simulated "Kitchen":



```
jetson@yahboom: ~
jetson@yahboom: ~    jetson@yahboom: ~    jetson@yahboom: ~
[asr-5] [INFO] [1750385414.028556399] [asr]: 0
[asr-5] [INFO] [1750385414.119921759] [asr]: 0
[asr-5] [INFO] [1750385414.210165105] [asr]: 0
[asr-5] [INFO] [1750385414.300471621] [asr]: 0
[asr-5] [INFO] [1750385414.390265068] [asr]: 0
[asr-5] [INFO] [1750385414.481523321] [asr]: 0
[asr-5] [INFO] [1750385414.571777356] [asr]: 0
[asr-5] [INFO] [1750385414.662565773] [asr]: 0
[asr-5] [INFO] [1750385414.754897462] [asr]: 0
{'load_data': '0.028', 'extract_feat': '0.036', 'forward': '4.403', 'batch_size': '1', 'rtf': '0.263'},
rtf_avg: 0.263: 100%|          | 1/1 [00:04<00:00,  4.40s/it]
rtf_avg: 0.263: 100%|          | 1/1 [00:04<00:00,  4.40s/it]
[asr-5] [INFO] [1750385419.211170831] [asr]: 请你先记住现在的位置然后依次导航去厨房和主卧室记住你所看到的物品最后返回出发时的位置告诉我你都在那两个地方看到了什么
[asr-5] [INFO] [1750385419.211922083] [asr]: okay😀, let me think for a moment...
[model_service-3] [INFO] [1750385423.231742756] [model_service]: The upcoming task to be carried out：1.调用记录当前位置函数记录位置2.导航去厨房，调用获取当前视角图像函数获取图像并记录看到的物品3.导航去主卧室，调用获取当前视角图像函数获取图像并记录看到的物品4.调用返回初始位置函数回到出发时的位置5.总结在厨房和主卧室看到的物品并告知用户
[model_service-3] [INFO] [1750385427.315891765] [model_service]: "action": ['get_current_pose()'], "response": 好的，我现在就记住现在的位置，就像记住了一个秘密基地一样。接下来，我要去厨房和主卧室探险啦！
[action_service-4] [INFO] [1750385436.995532898] [action_service]: Recorded Pose - Position: x=-0.07054017845186356, y=0.07985647398462059, z=0.0
[action_service-4] [INFO] [1750385436.996906727] [action_service]: Recorded Pose - Orientation: x=0.0, y=0.0, z=-0.3472589691926351, w=0.9377692724307342
[model_service-3] [INFO] [1750385441.388327622] [model_service]: "action": ['navigation(G)'], "response": 我已经记住了当前位置，现在准备导航去厨房。就像一个勇敢的探险家，出发！
```

The robot executes the following steps according to the output process of the decision layer:

```
[model_service-3] [INFO] [1750385441.388327622] [model_service]: "action": ['navigation(G)'], "response": 我已经记住了当前位置，现在准备导航去厨房。就像一个勇敢的探险家，出发！
[action_service-4] [INFO] [1750385449.596979417] [action_service]: navigation Goal accepted
[action_service-4] [INFO] [1750385516.629335533] [action_service]: Navigation succeeded!
[model_service-3] [INFO] [1750385520.996419381] [model_service]: "action": ['seewhat()'], "response": 我已经到达厨房了，现在开始用我的火眼金睛观察这里有什么物品。
[model_service-3] [INFO] [1750385537.403688769] [model_service]: "action": [], "response": 在厨房里，我看到了一支笔静静地躺在地上。看来这里除了烹饪工具，还藏着一些小秘密呢！接下来，我要去主卧室看看了。
[model_service-3] [INFO] [1750385555.608727072] [model_service]: "action": ['navigation(A)'], "response": 我已经记录了在厨房看到的物品，现在准备前往主卧室。就像一个好奇的侦探，继续我的探索之旅！
[action_service-4] [INFO] [1750385565.962718690] [action_service]: navigation Goal accepted
[action_service-4] [INFO] [1750385595.045950277] [action_service]: Navigation succeeded!
[model_service-3] [INFO] [1750385599.609402332] [model_service]: "action": ['seewhat()'], "response": 我已经到达主卧室了，现在开始仔细观察这里有什么物品。
[model_service-3] [INFO] [1750385611.106135108] [model_service]: "action": [], "response": 在主卧室里，我看到了一盒橙色包装的纸巾放在光滑的地面上。这个地方看起来很整洁，纸巾显得格外显眼。现在，我要返回出发时的位置了。
[model_service-3] [INFO] [1750385632.170785773] [model_service]: "action": ['navigation(zero)'], "response": 我已经记录了在主卧室看到的物品，现在准备返回出发时的位置。就像完成了一次小旅行，满载而归！
[action_service-4] [INFO] [1750385642.582316949] [action_service]: navigation Goal accepted
[action_service-4] [INFO] [1750385672.035012715] [action_service]: Navigation succeeded!
[model_service-3] [INFO] [1750385679.513325513] [model_service]: "action": [], "response": 我已经回到了出发时的位置。让我来总结一下我在厨房和主卧室看到的物品吧！在厨房，我看到了一支笔静静地躺在地上；而在主卧室，我则发现了一盒橙色包装的纸巾放在光滑的地面上。这两个地方各有特色，是不是很有趣呢？
[model_service-3] [INFO] [1750385706.296938429] [model_service]: "action": ['finishtask()'], "response": 我已经完成了全部任务，告诉了你在厨房和主卧室看到的物品。如果还有其他需要帮忙的地方，随时告诉我哦！
```

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
```

4.1 Case 1

action_service.py program:

In Case 1, the seewhat , **navigation** , **load_target_points** , and **get_current_pose** methods in **the CustomActionServer** class are used . **Seewhat** has been explained in **the Multimodal Visual Understanding + Robotic Arm Gripping** section. This section explains the two new functions: **navigation** , **load_target_points** , and **get_current_pose .**

In the **init_ros_comunication** initialization function, a nav2 navigation client is created to request the ros2 navigation action server for subsequent sending of navigation target point requests; a TF listener is created to monitor the coordinate transformation between map and basefootprint.

```
#Create a navigation function client and request the navigation action server
self . navclient = ActionClient ( self , NavigateToPose , 'navigate_to_pose' )
#Create tf listener to monitor coordinate transformation
self . tf_buffer = Buffer ()
self . tf_listener = TransformListener ( self . tf_buffer , self )
```

The load_target_points function is responsible for loading the target point coordinates in the map mapping file map_mapping.yaml and creating a navigation dictionary to store characters and corresponding map coordinates, where each point coordinate is PoseStamped type data.

```
def  load_target_points ( self ):
    with  open ( self . map_mapping_config , 'r' ) as  file :
        target_points = yaml . safe_load ( file )
    self . navpose_dict = {}
    for  name , data  in  target_points . items ():
        pose = PoseStamped ()
        pose . header . frame_id = 'map'
        pose . pose . position . x = data [ 'position' ] [ 'x' ]
        pose . pose . position . y = data [ 'position' ] [ 'y' ]
        pose . pose . orientation . x = data [ 'orientation' ] [ 'x' ]
        pose . pose . orientation . y = data [ 'orientation' ] [ 'y' ]
        pose . pose . orientation . z = data [ 'orientation' ][ 'z' ]
        pose . pose . orientation . w = data [ 'orientation' ] [ 'w' ]
        self . navpose_dict [ name ] = pose
```

Navigation function: Receives a character parameter (corresponding to the character in the above map mapping), parses the coordinates of the corresponding character from the dictionary, and uses the self.navclient navigation client object to request the navigation action server of ros2. When the navigation action server returns a value of 4, it means that the navigation is successful, and other values represent failures (possibly due to obstacles, planning failures, etc.). After the navigation is completed, the results of the action execution are fed back to the large model.

```
def  navigation ( self , point_name ):
    #Get the target point coordinates
    if  point_name == "zero" :
        target_pose = self . record_pose
    else :
        target_pose = self . navpose_dict . get ( point_name )
```

```
    goal_msg = NavigateToPose . Goal ()
    goal_msg . pose = target_pose
    send_goal_future = self . navclient . send_goal_async ( goal_msg )
    rclpy . spin_until_future_complete ( self , send_goal_future )

    goal_handle = send_goal_future . result ()
    if  not  goal_handle . accepted :
        self . get_logger () . info ( 'navigation Goal rejected' )
        return  False

    self . get_logger () . info ( 'navigation Goal accepted' )
    get_result_future = goal_handle . get_result_async ()
    rclpy . spin_until_future_complete ( self , get_result_future )

    result = get_result_future . result ()
    if  result . status == 4 :   # 4 indicates successful navigation
        self . get_logger () . info ( 'Navigation succeeded!' )
        self . action_status_pub ( f'Robot feedback: Execution 
navigation({point_name}) completed' )
    else :
        self . get_logger () . info ( f'Navigation failed with status: 
{result.status}' )
        self . action_status_pub ( f'Robot feedback: Failed to execute 
navigation({point_name})' )
```

The get_current_pose function is used to record the robot's current map coordinates in the global coordinate system map and put the coordinates into the dictionary for subsequent retrieval.

```
def  get_current_pose ( self ): #Record the current coordinates in record_pose
    '''
    Get the current position in the global map coordinate system
    '''
    # Get the current target point coordinates
    transform = self . tf_buffer . lookup_transform (
        'map' ,
        'base_footprint' ,
        rclpy . time . Time ())
    # Extract position and posture
    pose = PoseStamped ()
    pose . header . frame_id = 'map'
    pose . pose . position . x = transform . transform . translation . x
    pose . pose . position . y = transform . transform . translation . y
    pose . pose . position . z = 0.0
    pose . pose . orientation = transform . transform . rotation
    self . navpose_dict [ 'zero' ] = pose
    # Print the recorded coordinates
    position = pose . pose . position
    orientation = pose . pose . orientation
    self . get_logger (). info ( f'Recorded Pose - Position: x={position.x}, y=
{position.y}, z={position.z}' )
    self . get_logger (). info ( f'Recorded Pose - Orientation: x=
{orientation.x}, y={orientation.y}, z={orientation.z}, w={orientation.w}' )
    self . action_status_pub ( f'Robot feedback: get_current_pose() success' )
```