Multimodal visual understand+SLAM navigation(Text Version)

1. Course Content

1. Learn to use the robot's visual understanding combined with SLAM navigation.
2. Learn new key source code.

2. Preparation

2.1 Content Description

> This course uses the Raspberry Pi 5 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 **[01. Robot Configuration and Operation Guide] -- [5.Enter Docker (For JETSON Nano and RPi 5)]**. For Orin boards, simply open a terminal and enter the commands mentioned in this course.

📝 This example uses `model:"qwen/qwen2.5-vl-72b-instruct:free","qwen-vl-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 Large Model Basics Course] -- [5.Configure AI large model]** .

2.2 Configuring the Mapping File

For detailed information on the principles and concepts of map mapping, please refer to the course content in [2. AI Large Model Basics - 3. Embodied Intelligent Robot System Architecture].

For Raspberry Pi 5 and Jetson Nano controllers, you must first enter the Docker container. This is not necessary for the Orin board.

Use VNC to connect to the robot desktop. Open a terminal in Docker and enter the command.

```
ros2 launch yahboomcar_nav laser_bringup_launch.py
```

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

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

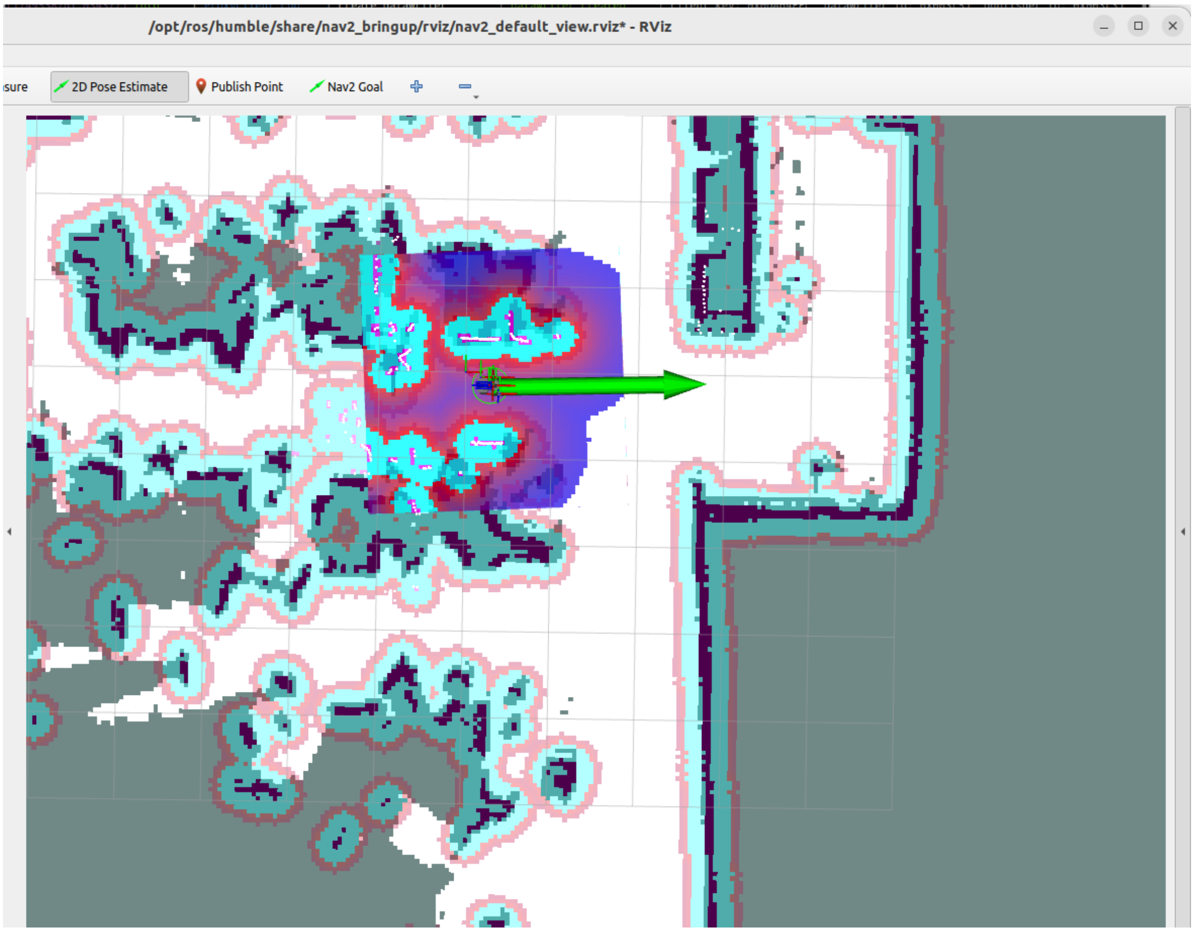
Wait for the navigation algorithm to start to display the map. Then open the Docker terminal and enter the command.

```
#Choose one of the two navigation algorithms (Raspberry Pi 5 and Jetson Nano only have standard navigation)
#Standard navigation
ros2 launch yahboomcar_nav navigation_teb_launch.py

#Fast relocalization navigation (Orin board)
ros2 launch yahboomcar_nav localization_imu_odom.launch.py ••use_rviz:=false load_state_filename:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream

ros2 launch yahboomcar_nav navigation_cartodwb_launch.py maps:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.yaml
```

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

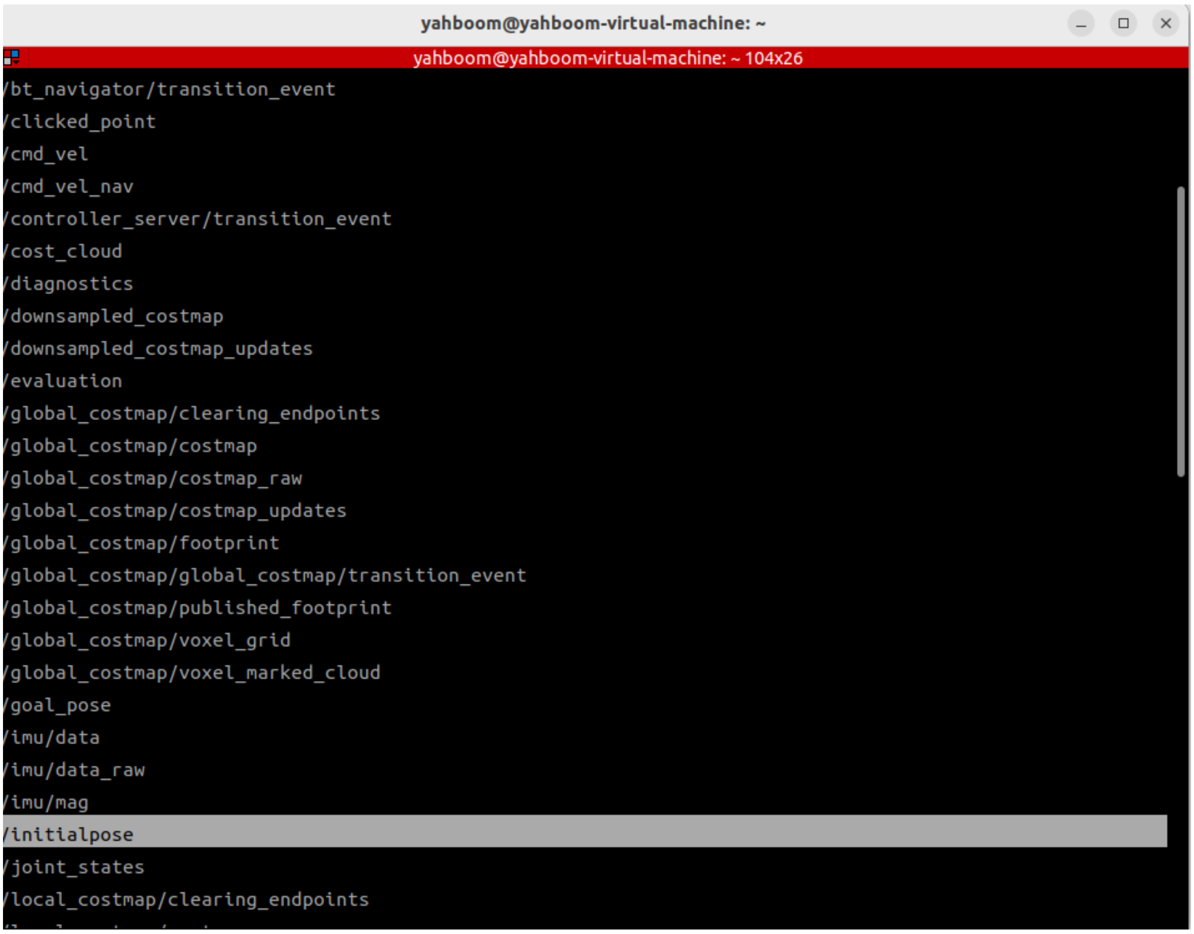
The map visualization interface will appear in rviz2. Click **2D Pose Estimate** in the upper toolbar to select it. Roughly mark the robot's position and orientation on the map.

The robot model will appear on the map as shown below:



Open a new terminal in the virtual machine and enter the command

```
ros2 topic list
```



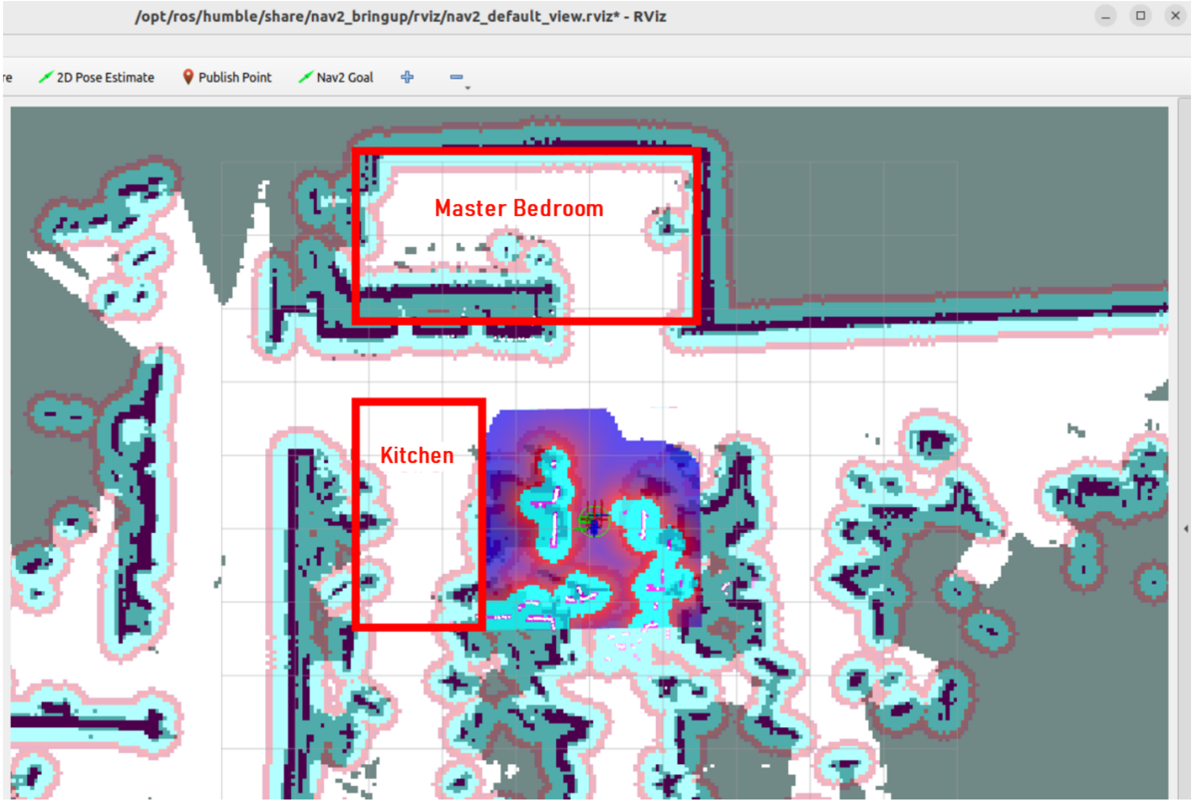

```
/bt_navigator/transition_event
/clicked_point
/cmd_vel
/cmd_vel_nav
/controller_server/transition_event
/cost_cloud
/diagnostics
/downsampled_costmap
/downsampled_costmap_updates
/evaluation
/global_costmap/clearing_endpoints
/global_costmap/costmap
/global_costmap/costmap_raw
/global_costmap/costmap_updates
/global_costmap/footprint
/global_costmap/global_costmap/transition_event
/global_costmap/published_footprint
/global_costmap/voxel_grid
/global_costmap/voxel_marked_cloud
/goal_pose
/imu/data
/imu/data_raw
/imu/mag
/initialpose
/joint_states
/local_costmap/clearing_endpoints
```

You can see the **/initialpose** topic. This topic is published by 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 and obtain the coordinates and orientation angle of a certain point in the map.

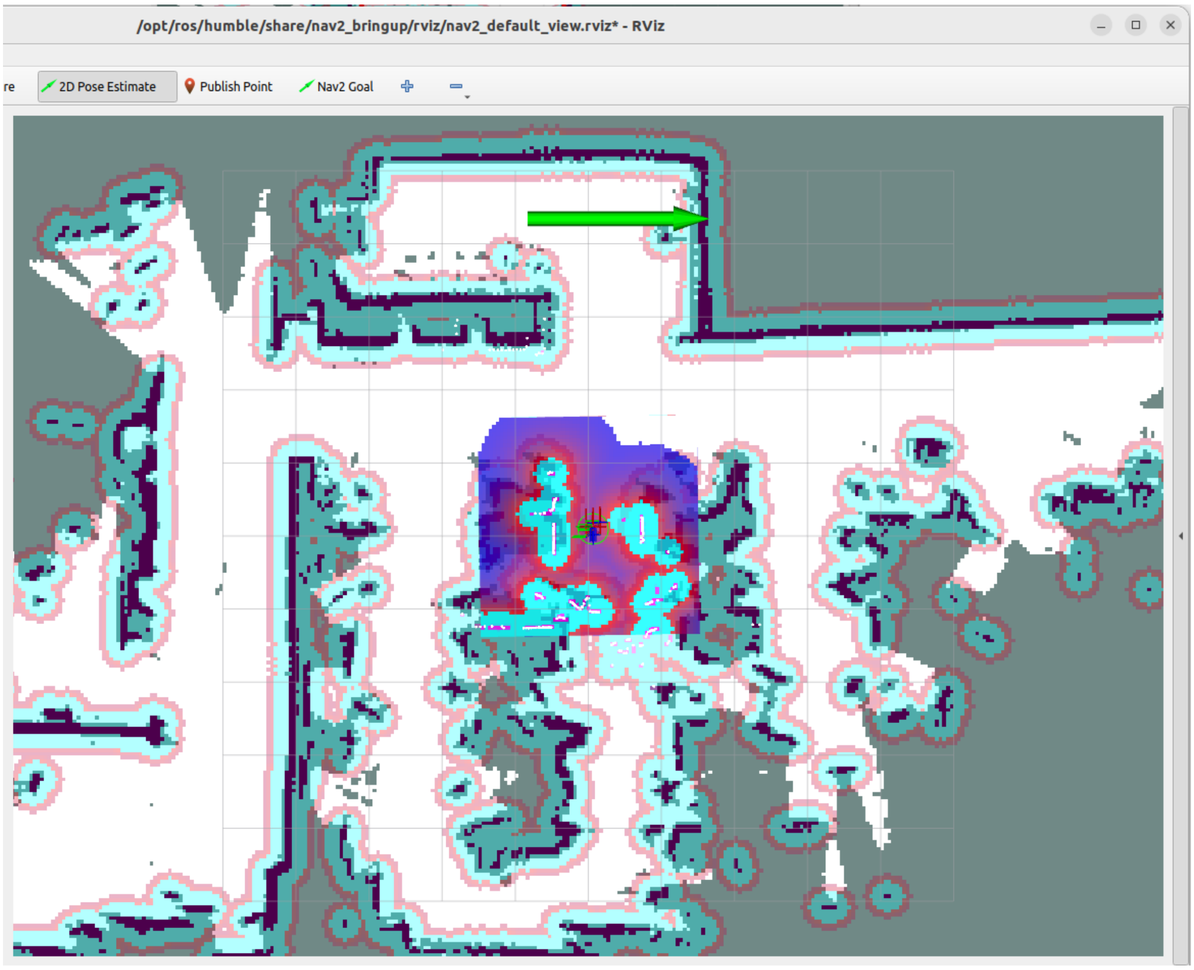
Enter the following command in the terminal to observe the data on the **/initialpose** topic.

```
ros2 topic echo /initialpose
```

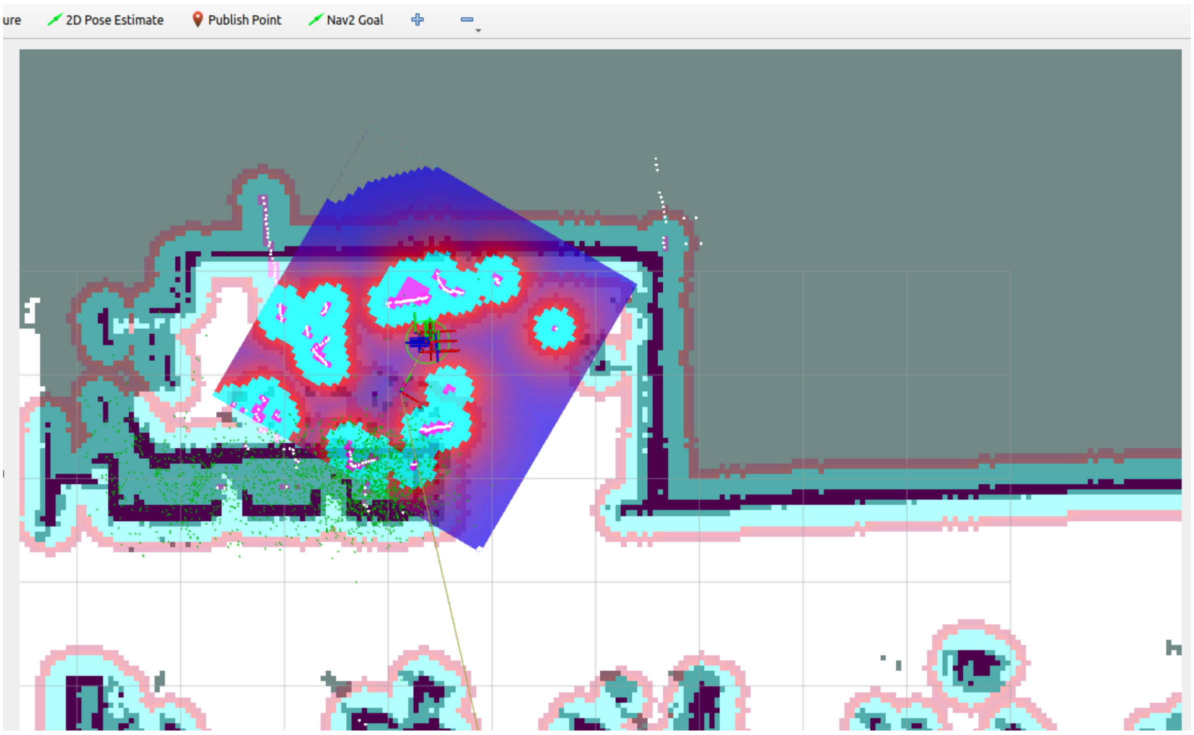
We can name any point in the map. Here we take "Master Bedroom" and "Kitchen" as examples.



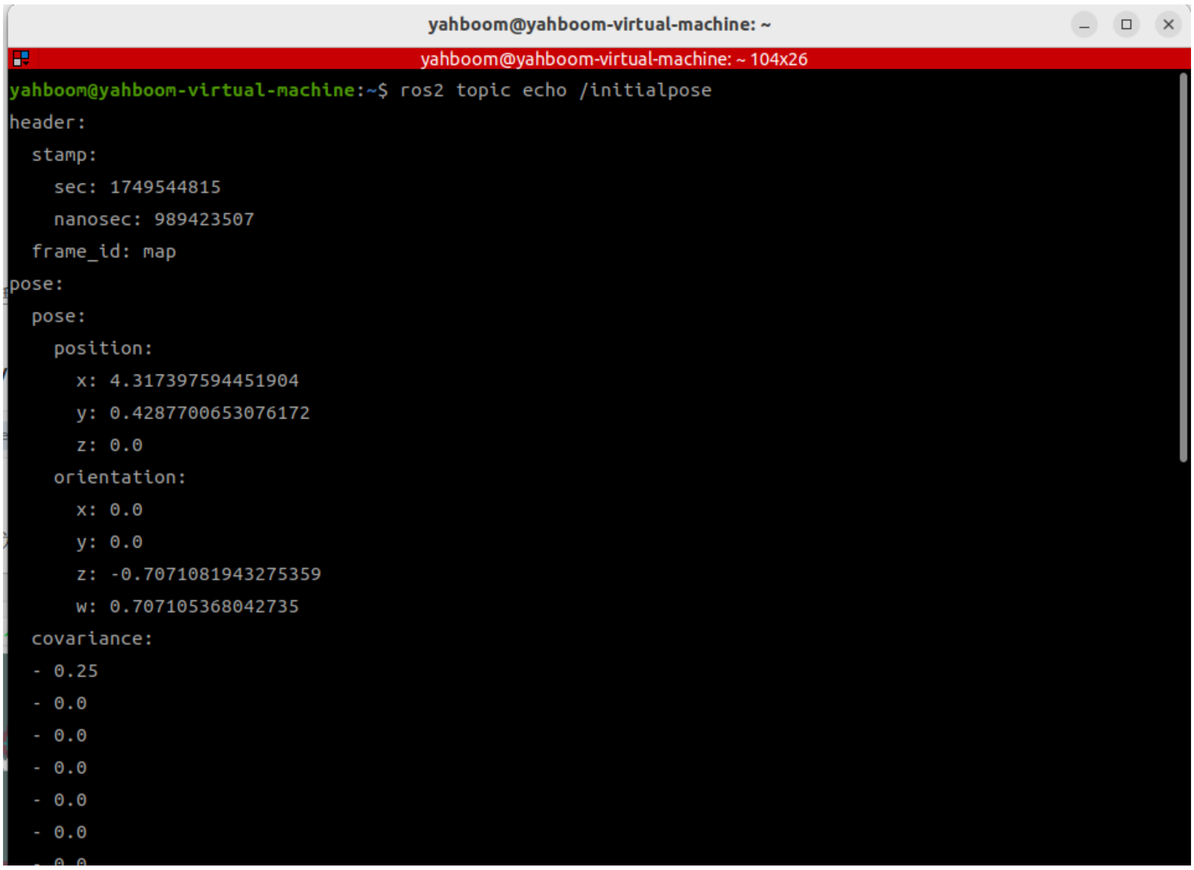
As shown in the figure below, we first click **2D Pose Estimate** tool, then select a place in the "Master Bedroom" area and click the left mouse button and hold it to adjust the direction. After confirming, release it.



The robot position will be adjusted to the position and direction just selected. The robot's estimated position and direction can be previewed 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.



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

jetson orin nano, jetson orin NX host:

```
/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/config/map_mapping.
yaml
```

jetson nano, Raspberry Pi host, you need to enter docker first ,

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/config/map_mapping.yaml
```

Replace "name" in A with "Master Bedroom" and "position" and "orientation" with the data we just saw in the **/initialpose** topic in the terminal.

```
#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: 1.6878514289855957
    y: -0.32615160942077637
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.4503881236302555
    w: 0.8928328724306793

#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

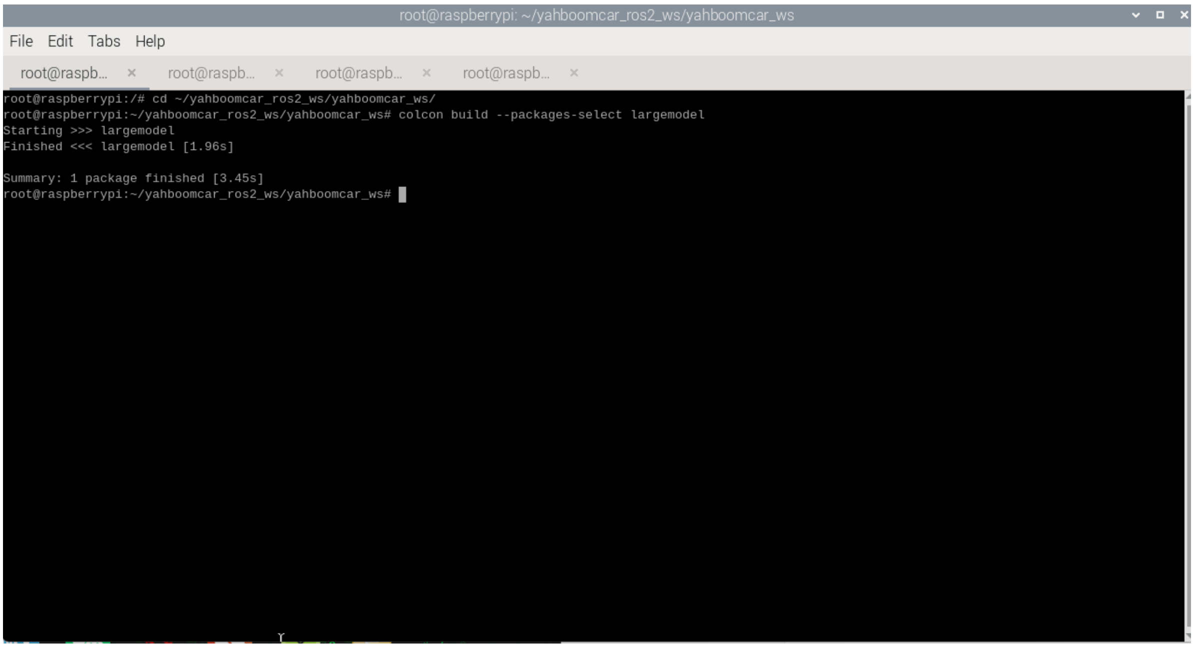
```
#根据实际的场景环境，自定义地图中的区域，可以添加任意个区域，注意和大模型的地图映射保持一致即可
#According to the actual scene environment, customize the areas in the map. You can add any number of areas, just make sure they are consistent with the map mapping of the large model
#地图映射Map mapping
A:
  name: 'Master Bedroom'
  position:
    x: 1.6878514289855957
    y: -0.32615160942077637
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.4503881236302555
    w: 0.8928328724306793
G:
  name: 'Kitchen'
  position:
    x: 2.367471694946289
    y: -1.568994164466858
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: -0.3106425066197889
    w: 0.950526818706855

#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 `/yahboomcar_ros2_ws/yahboomcar_ws#/` workspace in the terminal: Recompile the largemodel function package to take effect **(jetson nano, Raspberry Pi host, you need to enter docker first)**

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws/
```

Recompile the feature package:

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



3. Run Case

3.1 Starting the Program

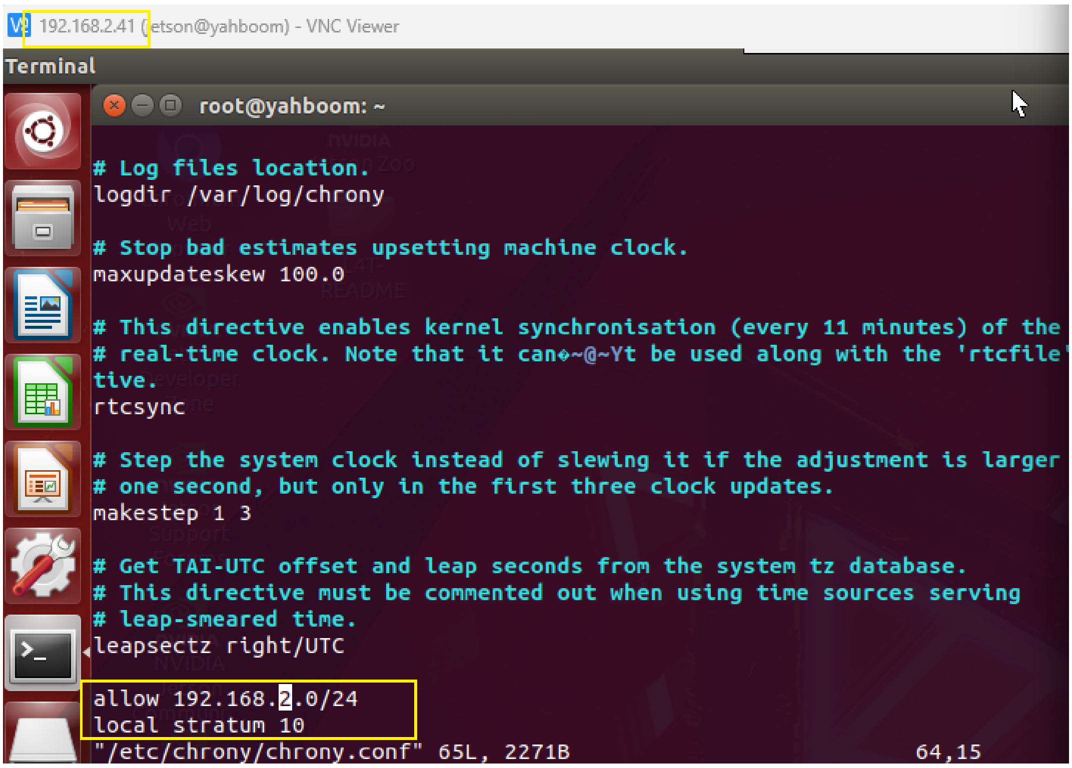
3.1.1 Jetson Nano board Startup Steps:

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. The following configuration is required:

- **Board Side (Requires Docker)**

Open a terminal and enter

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



Add the following two lines at the end of the file (192.168.2.0/24 is filled in according to the actual network segment. Here we take the current board IP address 192.168.2.41 as an example)

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

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

```
sudo service chrony restart
```

- **VM side**

Open a terminal and type

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

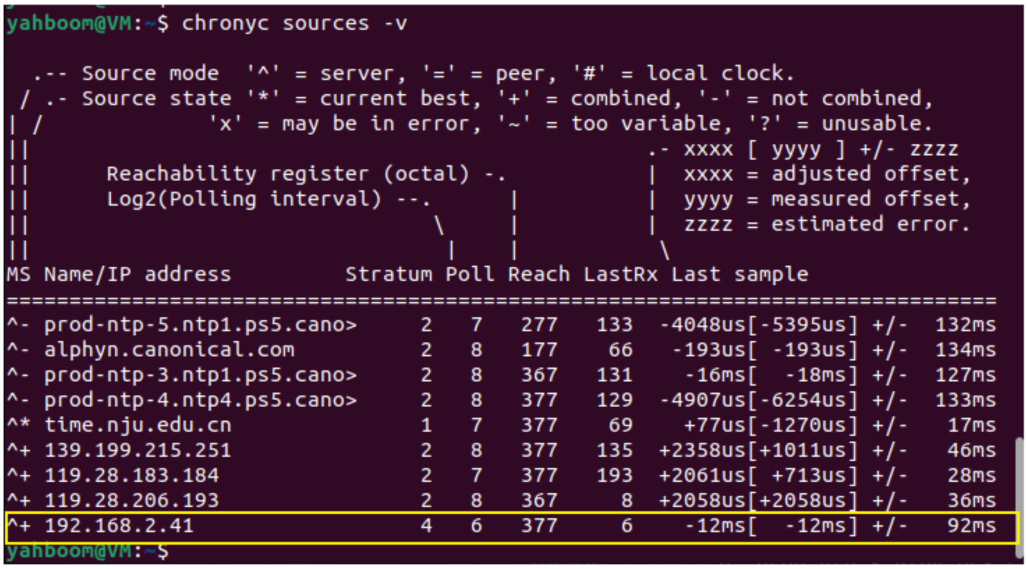
The 192.168.2.41 entry above is the board's IP address.

Enter the following command to take effect.

```
sudo chronyc reload sources
```

Then enter the following command to check the delay. If the IP address appears, it means it is normal.

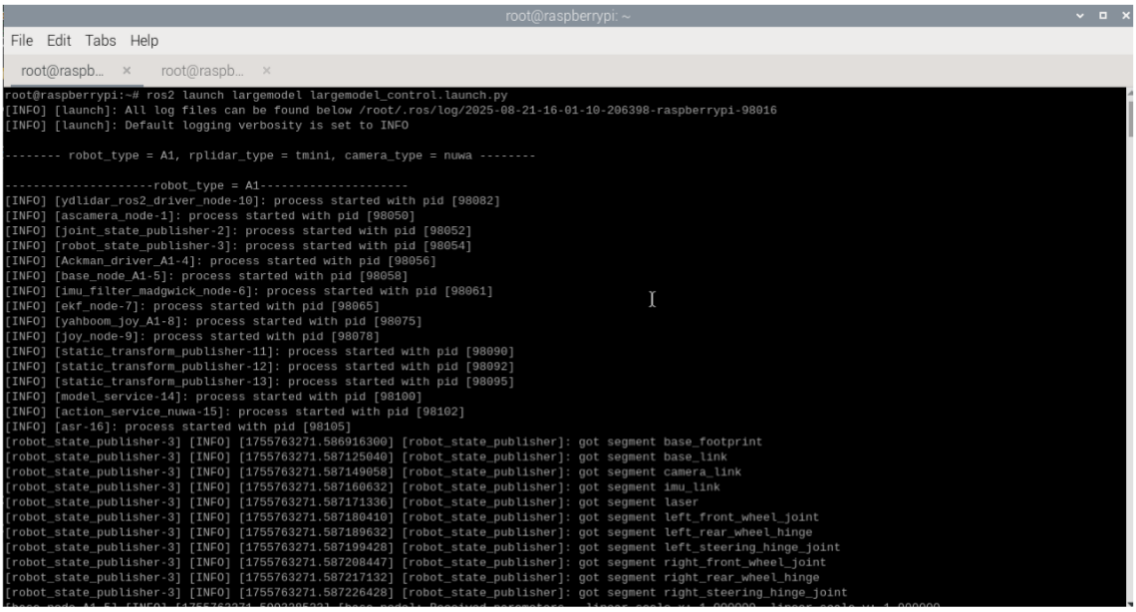
```
chronyc sources -v
```



- **Start the program**

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

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



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

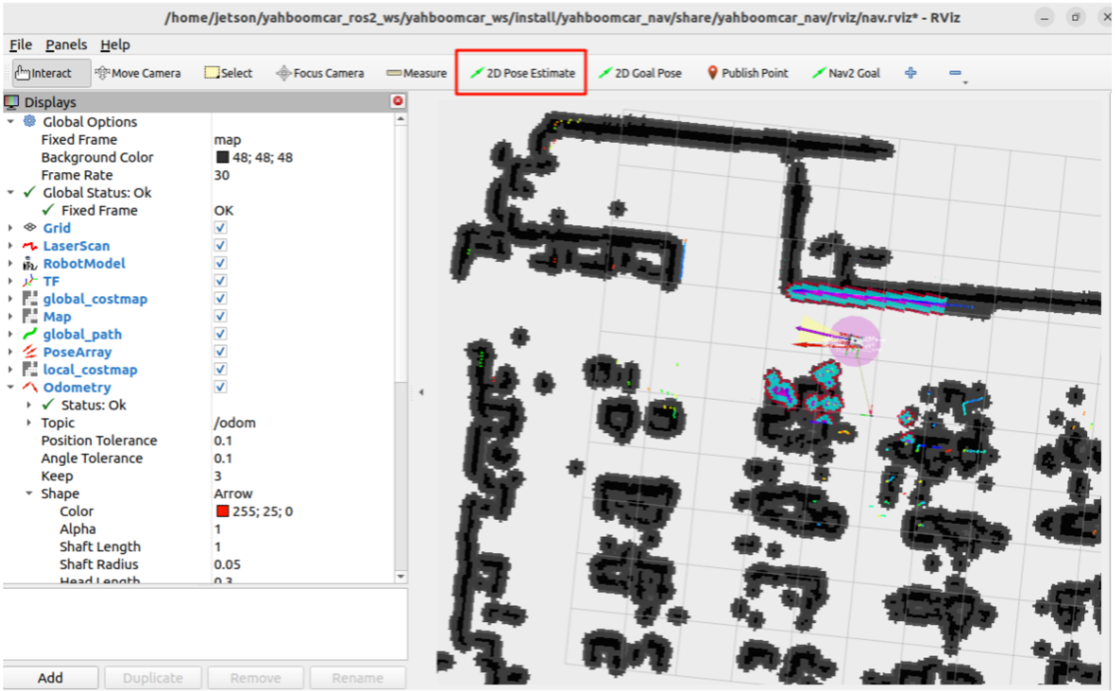
```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start and the map will be displayed. Then open a virtual machine 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 virtual machine to save the navigation map on the virtual machine.

Then 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.



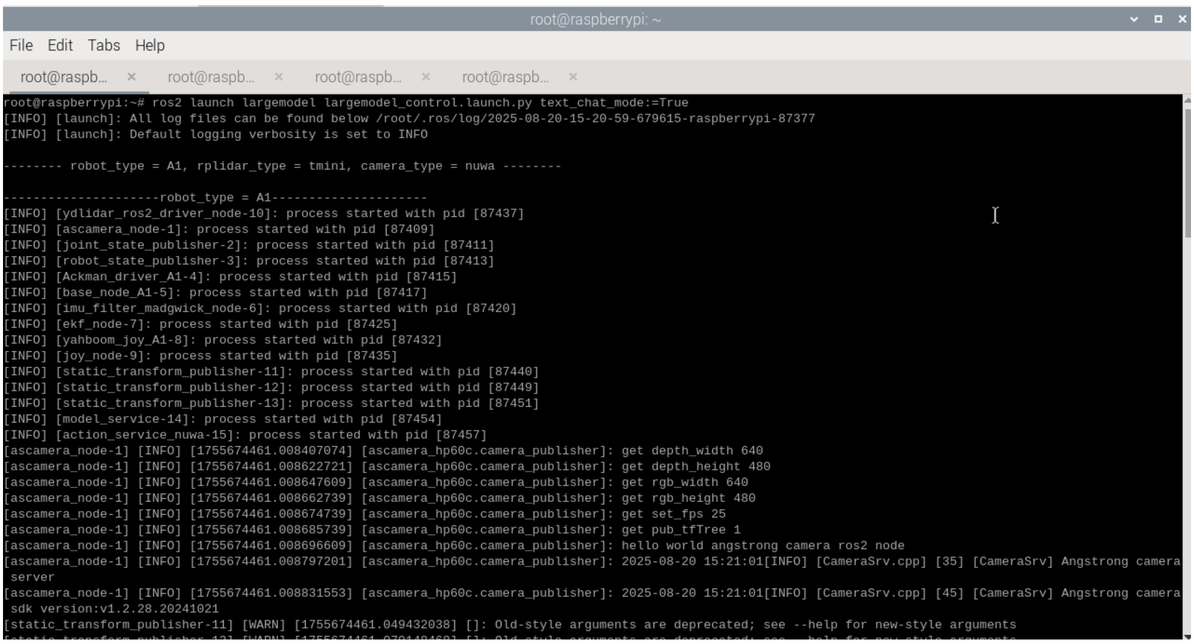
3.1.1 Raspberry Pi PI5 and ORIN board startup steps:

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

Open a terminal in Docker and enter the following command:

Open a terminal in Docker and enter the following command:

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



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

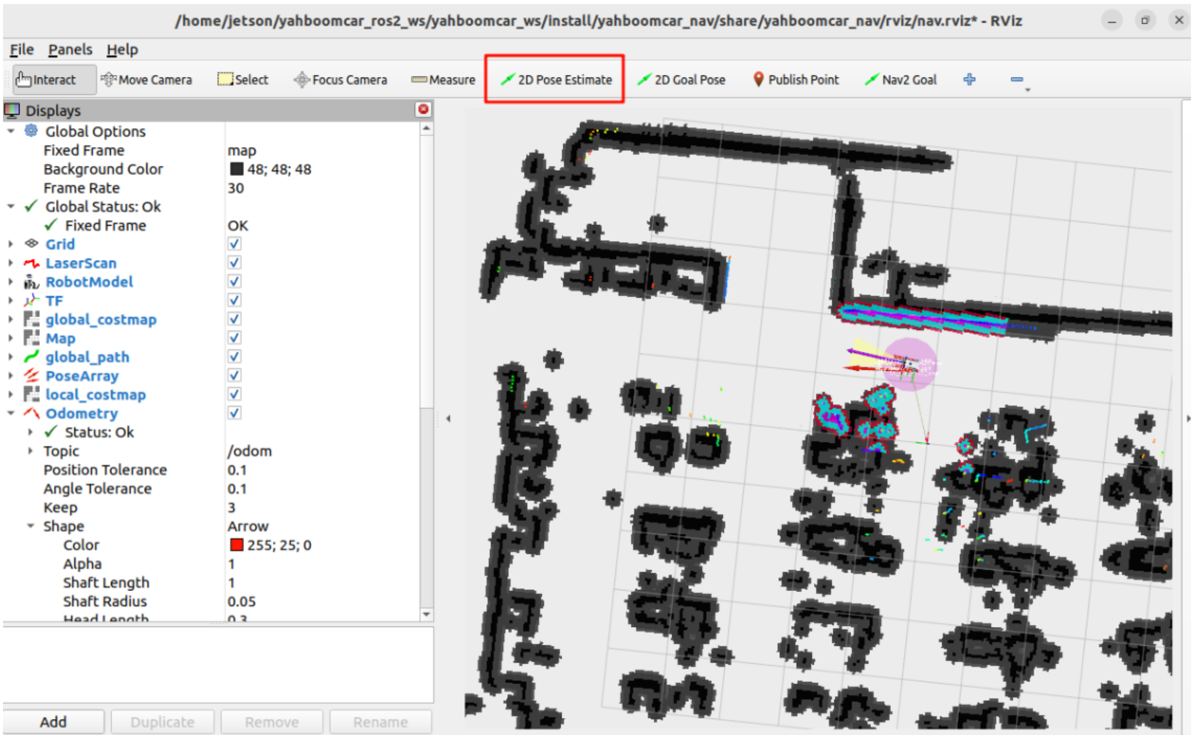
```
ros2 launch yahboomcar_nav display_nav_launch.py
```

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

```
#Choose one of the two navigation algorithms (Raspberry Pi 5 and Jetson Nano only use standard navigation)
#Standard navigation
ros2 launch yahboomcar_nav navigation_teb_launch.py

#Fast relocation navigation (Orin board)
ros2 launch yahboomcar_nav localization_imu_odom.launch.py use_rviz:=true load_state_filename:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream

ros2 launch yahboomcar_nav navigation_cartodwb_launch.py maps:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.yaml
```

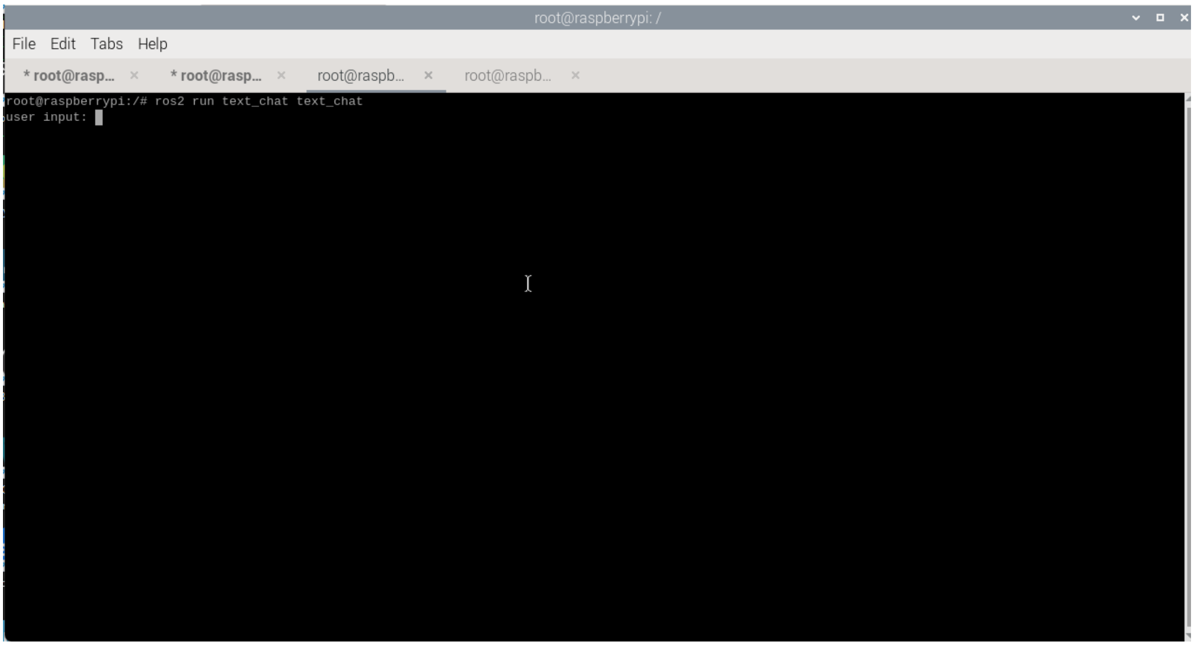
Note: yahboomcar.yaml and yahboomcar.pbstream must be mapped simultaneously, meaning they are the same map. Refer to the cartograph mapping algorithm to save the map.

After that, follow the navigation launch 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 initial positioning, preparations are complete.



Enter the same docker in multiple terminals and start it.

```
ros2 run text_chat text_chat
```



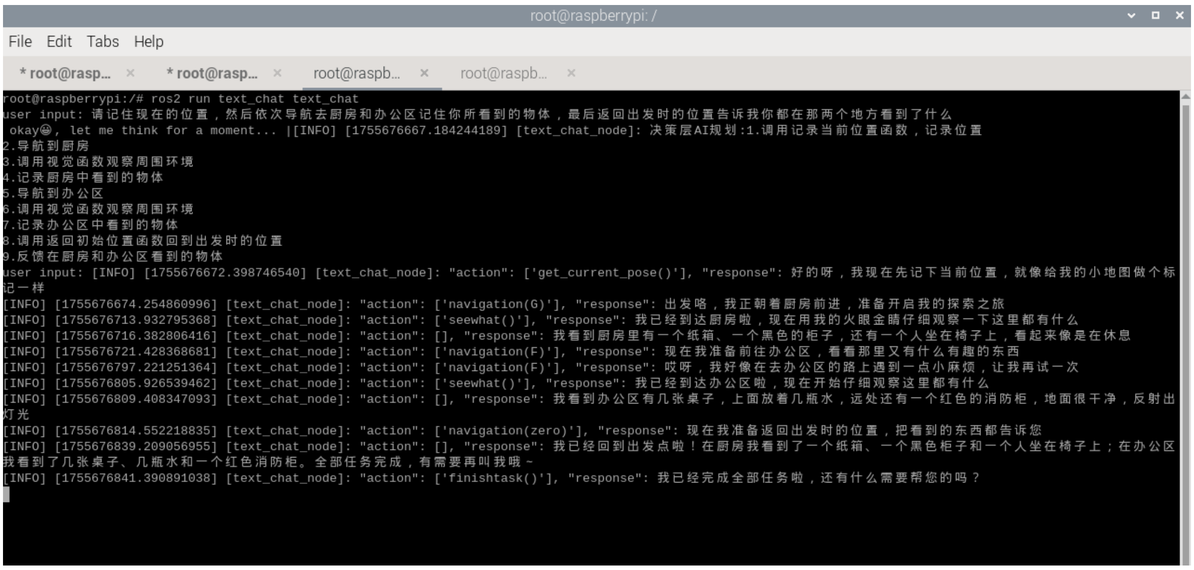
3.2 Test Cases

Here are some reference test cases; users are welcome to create their own dialogue commands.

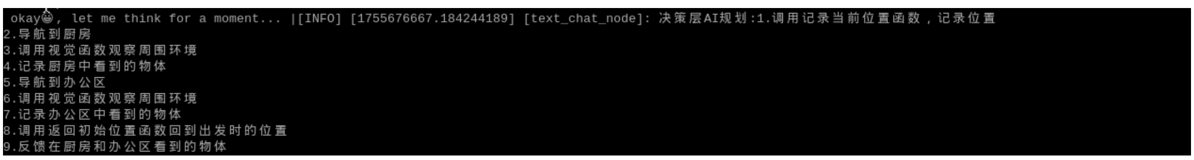
- Please remember your current location, then navigate to the kitchen and office area, remember what you see, and then return to your starting location. Tell me what you see in each of these two places.

3.2.1 Example 1

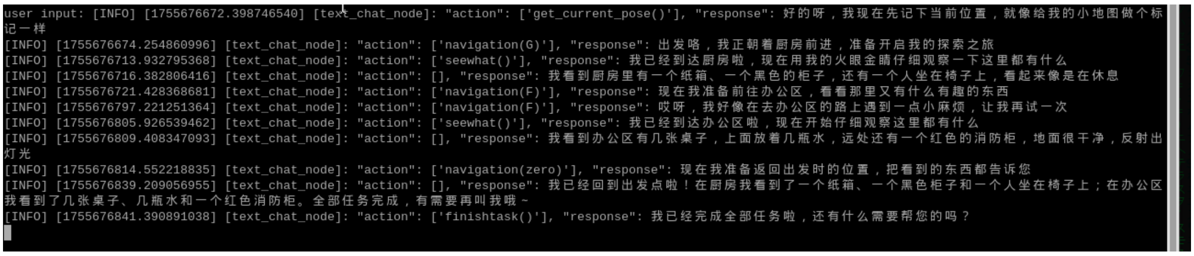
Type "Please remember your current location, then navigate to the kitchen and office area, remember what you see, and then return to your starting location. Tell me what you see in each of these two places?" in the terminal. The terminal will print the following message:



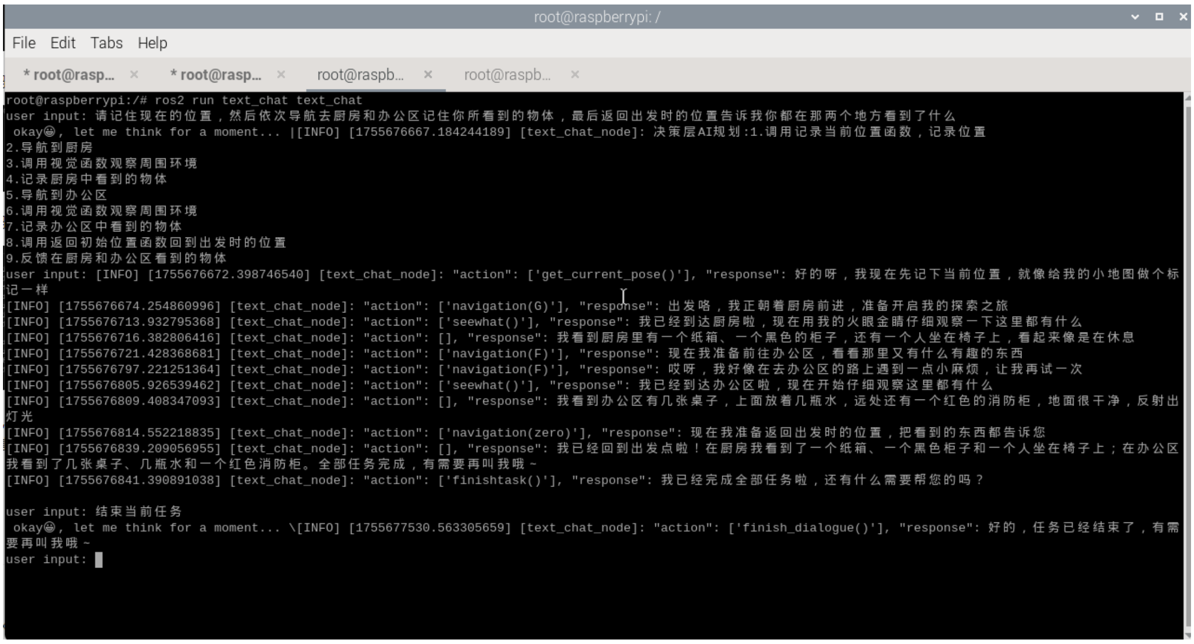
The decision-layer model outputs the planned task steps:



The execution-layer model then executes the task steps:



After completing the task, the robot enters a waiting state. Instructions from this state are directly passed to the execution-layer model, and all conversation history is retained. You can press the `ENTER` key in the terminal to continue the conversation and enter the "End current task" command to terminate the current task cycle and start a new one.



4. Source Code Parsing

Source code located at:

Jetson Orin Nano, Jetson Orin NX Host:

```
#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
```

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
```

4.1 Example 1

action_service.py:

Example 1 uses the **seewhat**, **navigation**, **load_target_points**, and **get_current_pose** methods in the **CustomActionServer** class. **seewhat** was already explained in the **Multimodal Visual Understanding** section. This section explains the newly introduced **navigation**, **load_target_points**, and **get_current_pose** functions.

In the **init_ros_comunication** initialization function, a nav2 navigation client is created to request the ROS2 navigation action server for subsequent navigation target point requests. A TF listener is created to monitor coordinate transformations between the map and basefootprint.

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

The **load_target_points** function loads the target point coordinates from the map mapping file, map_mapping.yaml, and creates a navigation dictionary to store characters and their corresponding map coordinates. Each point coordinate is a PoseStamped data type.

```
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.). self.nav_status indicates that the navigation flag can be interrupted through dialogue, and the result of the action execution is fed back to the large model after the navigation is completed.

```
 def navigation(self, point_name):
 """
    Get the target point coordinates from the navpose_dict dictionary and
navigate to the target point
    """
    self.nav_runing = True
    point_name = point_name.strip("'\"")
    if point_name not in self.navpose_dict:
        self.get_logger().error(
```

```
            f"Target point '{point_name}' does not exist in the navigation
dictionary."
        )
        self.action_status_pub("navigation_3", point_name=point_name)
        return

    if self.first_record:
        # 出发前记录当前在全局地图中的坐标(只有在每个任务周期的第一次执行时才会记录)/ before
starting a new task, record the current pose in the global map
        transform = self.tf_buffer.lookup_transform(
            "map", "base_footprint", rclpy.time.Time()
        )
        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
        self.first_record = False

    # Get the target point coordinates
    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)

    def goal_response_callback(future):
        goal_handle = future.result()
        if not goal_handle or not goal_handle.accepted:
            self.get_logger().error("Goal was rejected!")
            self.action_status_pub("navigation_1", point_name=point_name)
            return

        get_result_future = goal_handle.get_result_async()

        def result_callback(future_result):
            result = future_result.result()

            if self.nav_status:  #Interrupt sign
                self.nav_status = False
                self.action_status_pub("navigation_5", point_name=point_name)
                self.nav_runing = False

            else:
                if result.status == 4:
                    self.action_status_pub(
                        "navigation_2", point_name=point_name
                    )# 执行导航成功 /execute navigation success
                    self.nav_runing = False
                else:
                    self.get_logger().info(
                        f"Navigation failed with status: {result.status}"
                    )
                    self.action_status_pub(
                       "navigation_4", point_name=point_name
                    )# 执行导航失败 /execute_navigation_failed
                    self.nav_runing = False
```

```
        get_result_future.add_done_callback(result_callback)

    send_goal_future.add_done_callback(goal_response_callback)
```

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 a 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())
    # Extracting position and pose
    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 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'机器人反馈:get_current_pose()成功')
```