HW4: Mission Planning with ROS2

Mobile Manipulation 2025 - ShanghaiTech University

Due Date: Dec 25, 2025

Attention

Before starting this assignment, ensure you have ROS2 Iron installed on your Ubuntu 22.04 system (all the code has been tested in ROS2 Iron). You should also have a basic understanding of ROS2. If you're not prepared, please refer to the official ROS2 tutorials here and work your way up to the chapter "Implementing custom interfaces".

Mission Planning Overview

Mission planning is a key concept for intelligent mobile robots. It involves breaking down a high-level task into a sequence of executable steps. For example, if a human requests a robot to "fetch a cup of water from the corridor," the robot would need to plan a series of actions, such as:

1. Pick up a cup.
2. Navigate to the water cooler in the corridor.
3. Fill the cup with water.
4. Return to the human.
5. Hand the cup to the human.

This process of breaking down a high-level task into manageable subtasks and executing them is known as Mission Planning.

Your Task

For this assignment we are providing you with a detailed map in the osmAG [1] format, which includes rooms represented as areas, doors as passages, and additional furniture and objects to enable robots to interact with the environment. The map is stored in osm_parser/data/ under the file name osmAG.osm. It is an XML file. You can open it with an IDE to understand its structure, or you can visualize it using the JOSM software, as shown in Figure 1.

To simplify your workload, a ROS2 package called osm_parser is provided:
https://robotics.shanghaitech.edu.cn/gitlab/moma2025/moma_hw4_packages
This package processes the map data from osmAG.osm into various OSMData classes.

In this assignment, you are required to accomplish one of the following weather-aware missions (randomly selected each run):

1. Cook & Clean Mission: cook a turkey in the oven, place the cooked turkey back on the dining table, wait for the oven to cool down, and call the dedicated cleaning service so that the oven returns to a clean state.
2. Wash & Store Mission: wash a dirty jacket in the washing machine, then follow a weather-dependent branch. When the random weather parameter is rain, you must load the jacket into a dryer, run the drying cycle, retrieve it, and store it inside the wardrobe. When the weather is sunny, you must hang the jacket on the clothesline rod until it becomes dry, then place it in the wardrobe. In both cases the wardrobe must end closed for the mission to finish.

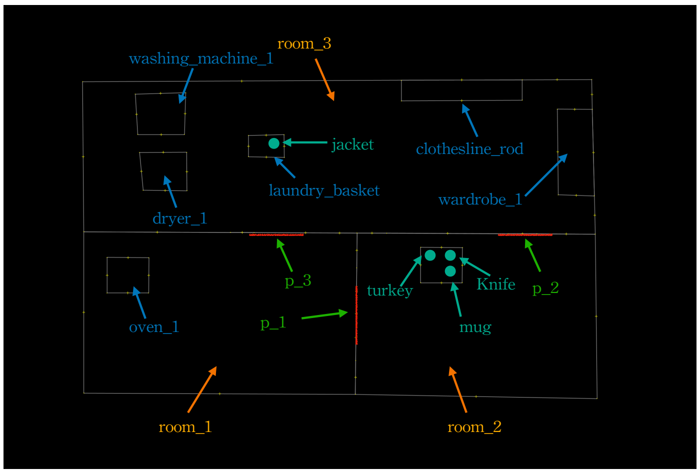
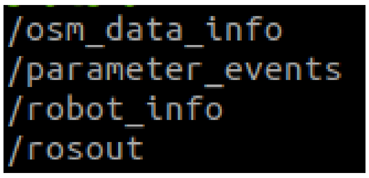


Figure 1: Visualization of OSMAG map in JOSM software



```
/osm_data_info
/parameter_events
/robot_info
/rosout
```

Figure 2: topic list

The simulator publishes a weather_info topic and exposes the parameter weather_condition so that your planner can discover whether the current run requires the dryer path or the outdoor drying path.

To simplify your workload, a ROS2 package called osm_parser has been provided. This package processes the map data from osmAG.osm into various OSMData classes. You can compile and run the package to start the simulator:

```
colcon build
source install/setup.bash
ros2 launch osm_parser start_sim.launch.py
```

All the map information will be loaded into memory. When running the simulator for the first time, all initial information from osmAG will be displayed in the CLI. Additionally, all relevant information in osmAG that can change state based on manipulation (such as passages, furniture, items) and the current state of the robot will be published via topics. You can subscribe to the relevant topics (osm_data_info, robot_info) to receive real-time information. (Hint: Your method should subscribe to these topics to automatically handle various random scenarios.)

In addition, we use a node to randomly select between two missions, ”cook_turkey” or ”wash_jacket,” and publish the selected mission as a parameter in ROS2 with a 50% probability for each option. You can use the following cmd to get the selected mission in this run:

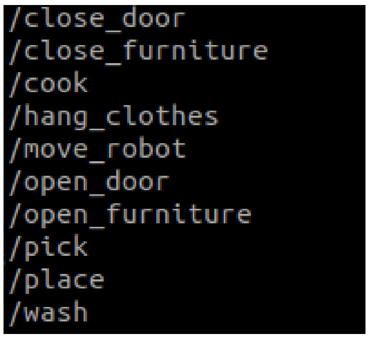
```
ros2 param get /random_mission_node selected_mission
```

(Hint: your method shall somehow use the parameter to automatically perform one of these two missions.)

Robot Services

In this assignment, you'll be focusing on high-level Mission Planning logic. We have abstracted the robot's actions into ROS2 Services that you can call. Ideally, your method should create a sequence of subtasks to achieve the high-level mission. While the Simulator is running, to see the available useful services, you shall check ROS document to learn how to see them.

Furthermore, to explore advanced service commands, you can use:



```
/close_door
/close_furniture
/cook
/hang_clothes
/move_robot
/open_door
/open_furniture
/pick
/place
/wash
```

Figure 3: service samples

ros2 service -h

including checking custom interface types. (Hint: Which could be helpful for your task.)

Service Rules

Each service follows specific rules. Your mission planning solution should adhere to these rules when calling services:

1. The robot will spawn randomly in one of the rooms: room$_{1,2,3}$. Your solution should adapt to these random scenarios.
2. The service open_door has a 50% chance of failure (as manipulation in the real world is often imperfect).
3. The robot has only one manipulator, so it cannot open a manual door when carrying an item. However, it can still open furniture and close doors without any issues.
4. The robot cannot pass through a passage unless the passage is open.
5. A passage can only be opened if its type is handle door, and the robot must be in one of the connected rooms to call open_door or close_door.
6. The robot cannot open or close furniture unless it is in the same room as the furniture.
7. Only furniture of type oven can be used for the cook action, and cooking requires the food to be inside a closed oven.
8. After cooking is complete, the oven becomes dirty and starts a cooldown timer. The new clean_furniture service succeeds only when the oven is closed, the cooldown time has elapsed, and the robot is at the oven's location.
9. The robot cannot Pick or Place an item unless it is in the same room as the furniture on which the object is placed, and the furniture is open.
10. Only furniture of type washing_machine can be used for the wash action, which also requires the machine to be closed with the clothes already inside.
11. When the weather is rain, clothes must be placed inside a dryer and the dry_clothes service must run to completion before the item can be stored. When the weather is sunny, the robot must hang the clothes on the clothesline until they dry before moving them to the wardrobe.
12. The wardrobe must be closed with the target item stored inside to satisfy the wash_jacket mission; the oven must be cleaned after cooking to satisfy the cook_turkey mission.

When the robot successfully completes the mission, the terminal will output:

Congratulations, You have completed the mission_1/2 successfully!

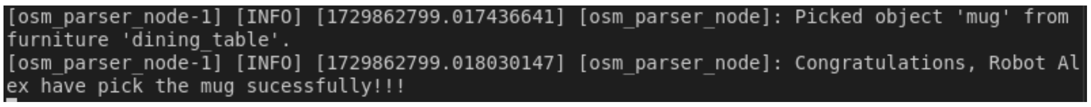
This confirms that your solution is correct.

Example

To simplify the assignment, we have provided a simple mission example: let the robot pick up the mug on the dining table. This example is straightforward, requiring only a call to the service: pick(mug, dining_table). For details, please refer to osm_parser/client_demo_node.py, which demonstrates a ROS2 client node implementation for calling a service, you can run this in cmd while the simulator is on:

```
ros2 run osm_parser client_demo_node
```

and can see the output, shown in Figure 3



```
[osm_parser_node-1] [INFO] [1729862799.017436641] [osm_parser_node]: Picked object 'mug' from
furniture 'dining_table'.
[osm_parser_node-1] [INFO] [1729862799.018030147] [osm_parser_node]: Congratulations, Robot Al
ex have pick the mug sucessfully!!!
```

Figure 4: Demo output

Hint:Your task is to achieve the same service call using one of the four methods mentioned in Section 'Submission Guidelines'.

We outline the basic decision criteria in this demo:

- The robot starts in room_2, and as seen on the map, the dining table is also in room_2, so the move_robot service is not needed.
- The dining table's status is open (as a table is, of course, not closed), so the open_furniture service is also unnecessary.
- Simply call the pick service.

Submission Guidelines

This assignment will be done in groups of four. Each member of the group will choose one of the following methods to implement the mission planning:

- State Machine (See SMACC2 — other libraries are allowed)
- Behavior Tree (See BehaviorTree.CPP)
- Symbolic Planning(See plansys2)
- LLM API

Each group's member should submit the respective files to GitLab individually according to the chosen method:

- For State Machine:
  Submit your state machine implementation in the relevant directory. (as a ROS2 package) Include a README.md explaining your approach and the reasoning behind your design choices, and a launch file to start your package all at once.
- For Behavior Tree:
  Submit your behavior tree implementation with all relevant files(as a ROS2 package) Include a README.md on how the tree is structured and any key decision points in the mission, and a launch file to start your package all at once.
- For Symbolic Planning:
  Provide your symbolic planner(as a ROS2 package) along with a README.md of how symbols and actions are mapped to the ROS2 services, and a launch file to start your package all at once.
- For LLM API:
  Please demonstrate your demo by either uploading a video or scheduling an appointment with the TA to present your results. Use MCP to let the LLM call the ROS services.

  Please do your best - especially for the LLM part we will grade generously. We want to see you try - even if the result may not be perfect.

Grading

The grade is calculated like this: 25% of your grade comes from the performance of your part. The rest of the 75% is the average grade of all 4 parts. We encourage collaboration within groups. While each of you will be using different methods, the end goal is the same, so feel free to share ideas and assist each other.

Good luck, and happy coding!

References

[1] Delin Feng, Chengqian Li, Yongqi Zhang, Chen Yu, and Sören Schwertfeger. Osmag: Hierarchical semantic topometric area graph maps in the osm format for mobile robotics. In 2023 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 1–7. IEEE, 2023.