HW4: Mission Planning with ROS2

Mobile Manipulation 2024 - ShanghaiTech University

Due: Dec 12 2024 (Selection due: Dec 1st)

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 will provide you with a detailed map using **osmAG** [1], 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's 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.

In this assignment, you are required to accomplish one of those missions:

1. Cook a turkey using the oven and place the cooked turkey back on the dining table
2. Wash a dirty jacket using the washing machine and hang the clean jacket on the clothesline rod.

To simplify your workload, a ROS2 package called `osm_parser` is provided:
`https://robotics.shanghaitech.edu.cn/gitlab/moma2024/hw/moma_hw4_packages`
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
```

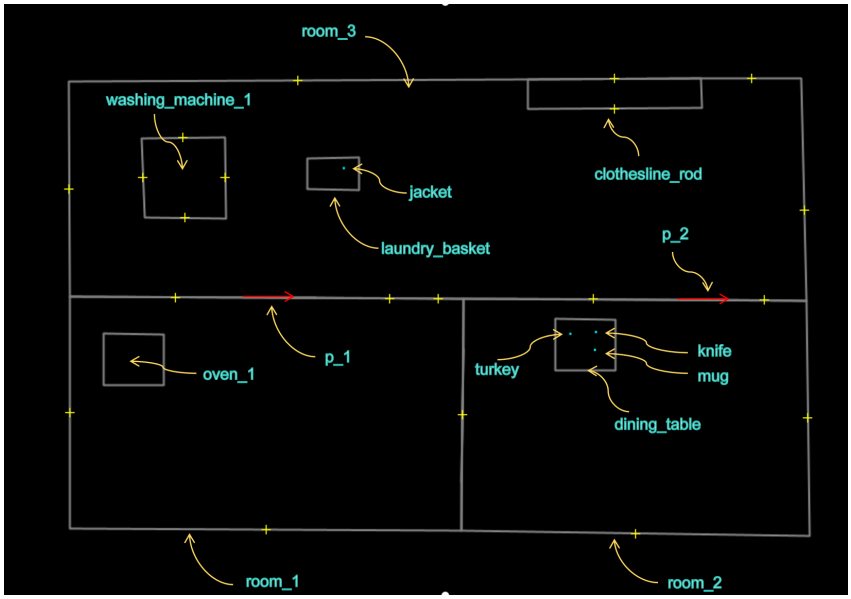

Figure 1: Visualization of OSMAG map in JOSM software

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

Figure 2: topic list

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 can use the command to expose them, Shown in Figure 2:

```
ros2 service list | grep -v "osm_parser_node\|random_mission_node"
```

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

```
ros2 service -h
```

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

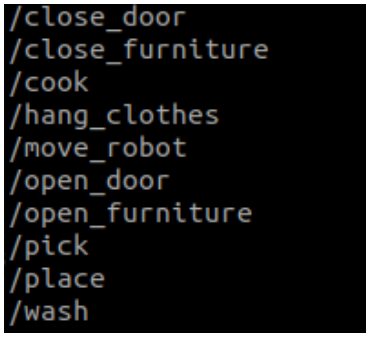

Figure 3: service list

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: $\text{room}_{1,2,3}$. 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, to reduce task difficulty, 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 `door`.
6. The robot can open or close a passage only if it is located in one of the rooms connected by that passage.
7. The robot cannot open or close furniture unless it is in the same room as the furniture.
8. Only furniture of type `oven` can be used for the `cook` action.
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.
11. The `cook` action can only be executed after food has been placed in the oven, and the oven has been closed.
12. The `wash` action can only be executed after clothes have been placed in the washing machine, and the machine has been closed.

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**

One team member should send the members of the HW4 group and who does what part to the TA, latest by Dec. 1!

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.

  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.

Summary

So, to sum up, you'll write 4 programs that execute a sequence of ROS2 services to achive one of two goals (cooed turkey on table; clean jacket on cloethesline). You can use the provided library to read the osmAG, but it is also fine to just hardcode the layout and tasks in your solutions (e.g. in the FSM strucutre; as pre-defined domain and problem files, as pre-defined LLM promot). Your solution should subscribe to the perception topics from the simulator and plan/ react/ re-plan accordingly.

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.