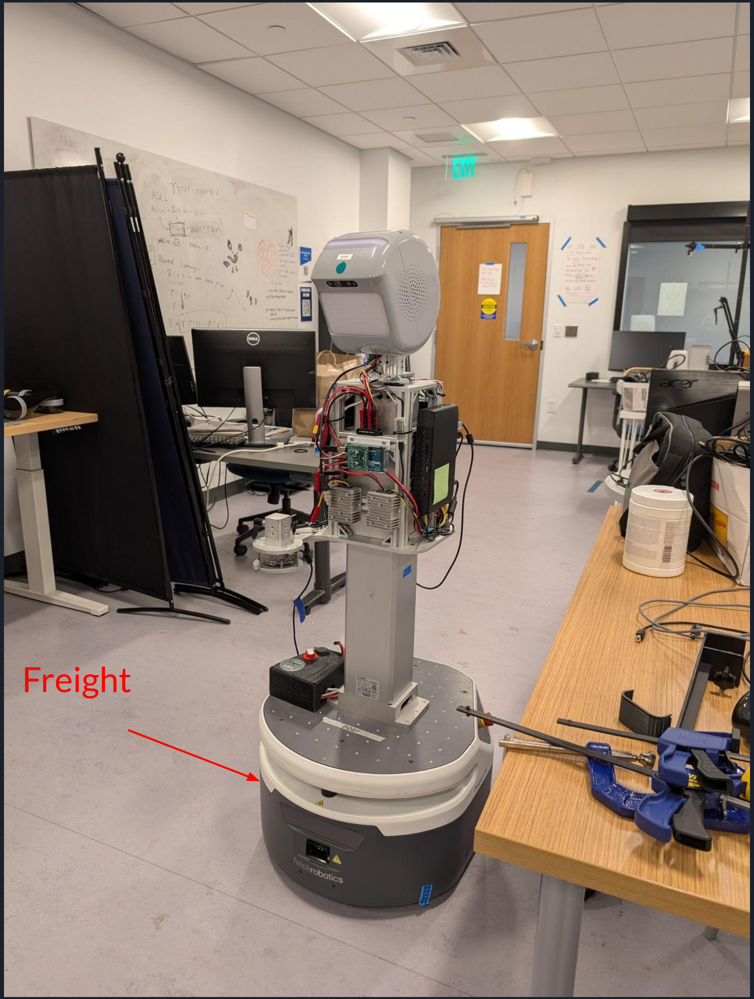
Freight Navigation

Project Goal

- The goal of this project was to create a robot that would be able to guide a user to a requested location on the 4th floor of the JCC.
- We used the mobile base of the Fetch robot named Boop.
- This mobile base is also known as a Freight robot.
- This robot includes a LIDAR which we used for obstacle detection, localization, and mapping.



VNC Access

Problem

- The version of linux was too old for the ssh plugin of VSCode to work.
- This meant that if we ever wanted to program the robot, we would need to hook it up to a monitor.
- Additionally, even if the version of linux was newer, any form of visualization window would not work.

Our Solution

- We set up a vncserver on the robot that one could connect to for a virtual desktop of the robot.
- Instead of programming in VSCode using the ssh plugin, we programmed in VSCode in the virtual desktop.
- Additionally, this allowed for visualization windows to function.

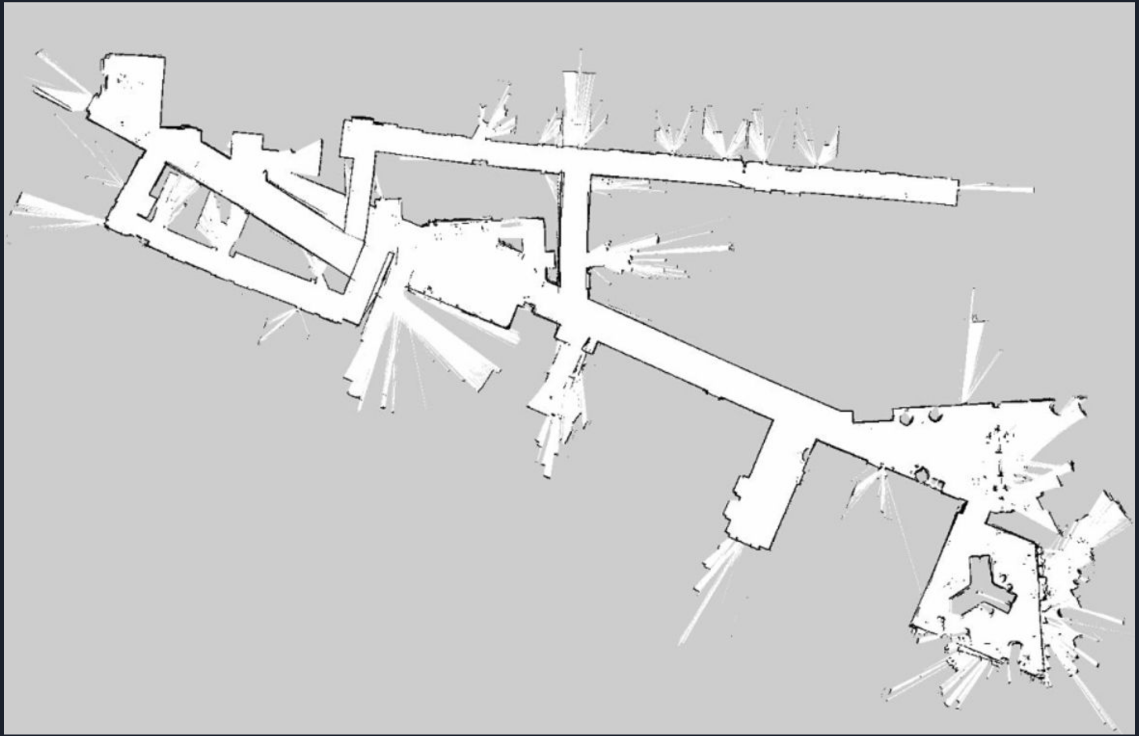
Mapping

- We initially mapped the 4th floor by driving the robot through the floor, starting at the AABL lab.
    - We did not have a strategy the first time around, and as a result we ended up with doubled mapped areas and an incorrect floor shape.
- When we remapped the floor to fix this issue, we had the strategy of starting on one of the ends of the floor, and avoided going to areas multiple times.
    - By not ending at the place we started, we avoided the problem of the mapping script closing the map prematurely, which was what led to the incorrect floor shape we had with the first incarnation.

Localization

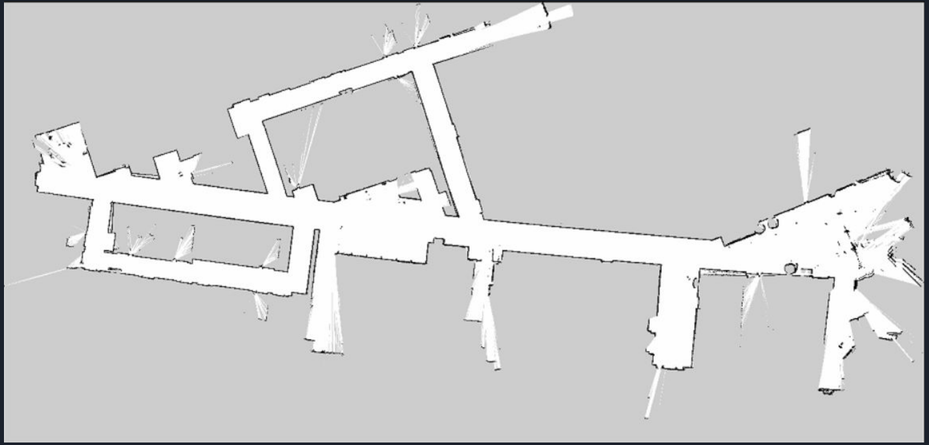
- The freight uses amcl, a Monte Carlo localization algorithm, to localize itself using the maps that the mapping script creates.
    - Black pixels are used to identify walls or other obstacles, and white is open space.
- Because modifying the map image itself also affects localization, a keepout map is used to keep the robot out of areas we don’t want it to go through.
    - Essentially this creates walls that do not affect localization.
    - This is how we keep the robot away from stairs, mark windows that the mapping script did not register as walls, and prevent the planning algorithm from creating a path through unmapped areas.

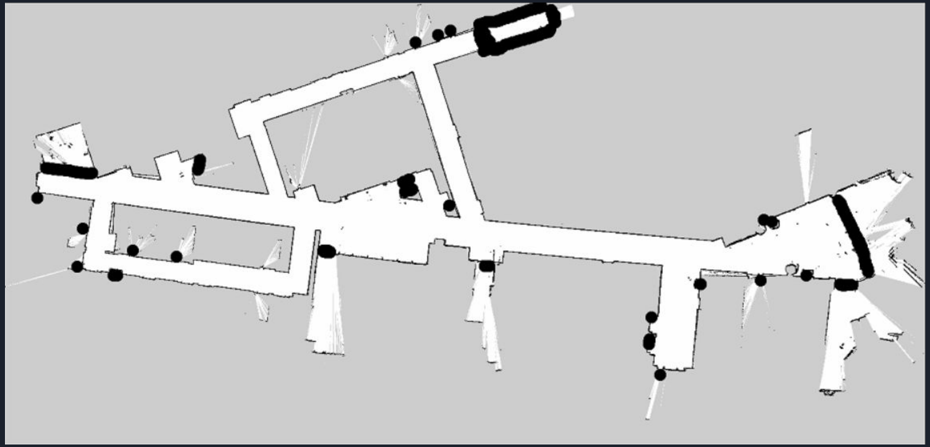
JCC 4th Floor Original Map



First Mapping Attempt

JCC 4th Floor Maps


Base Map


Keepout Map

Local Path Planning

Problem

- The default local path planner the freight uses struggled with corners.
- It would often get stuck on the corner in at least one direction.
- This can lead to the robot stripping its wheels if it gets stuck while on a plastic lip.

Our Solution

- We changed from the default local path planner to teb local path planner.
- This planner does not struggle with corners at all due to trying to stay in the center of hallways at all times.
- Additionally, it is better at obstacle avoidance.
- The main downside of this planner is that it is computationally expensive.

TEB Local Path Planner Optimization

- With the default parameters and our large map, adding TEB led to our robot jittering.
- This jittering is caused by the computation time at each step of planning taking too long.
- In order to speed up the computation time, we had to increase the local costmap resolution.
- Additionally, we had to adjust the inflation distance of the robot and obstacles, as the robot was essentially trying to walk on a tightrope when staying in the center of a hallway.

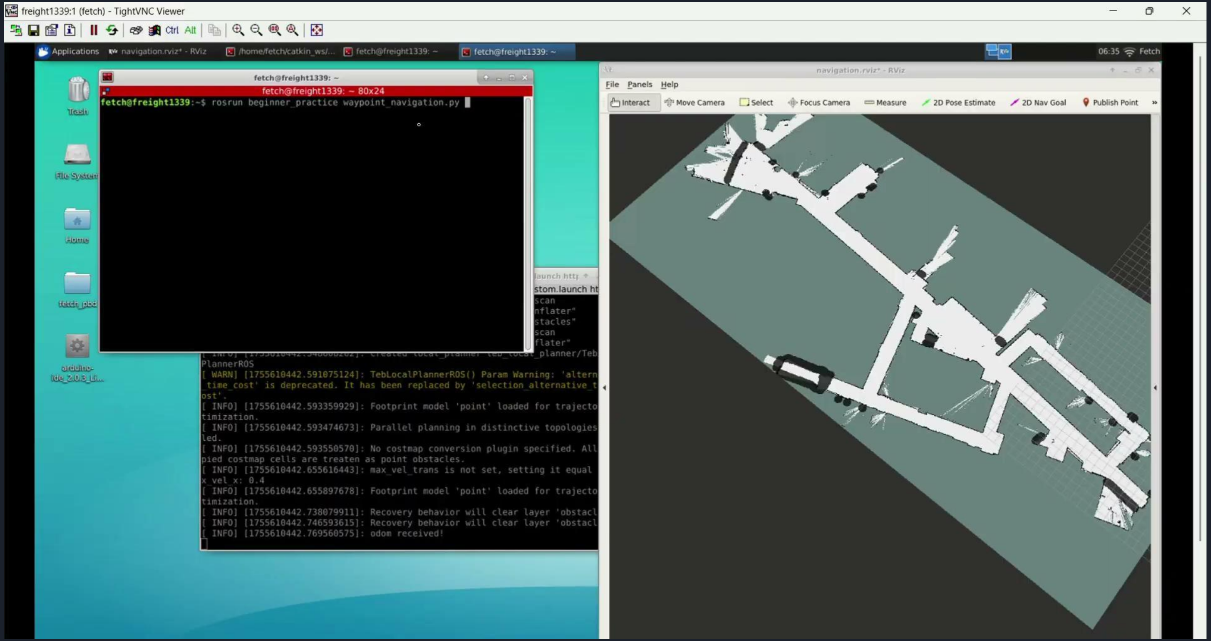
Navigation Program

- Test Program
  - First we made a small program to have the robot move in a square just to test how we would move it in the first place.
  - We used a publisher that would write to the “cmd_vel” topic with a message having the robot move forward and then turn 90 degrees.
    - This program seemed fine on the surface but it never moved and the problem was that without any kind of rospy.sleep() the publisher was never actually created
  - Once that was resolved the robot was able to move in a “square” pattern.
    - The issue here is that the wheels are variable and sometimes the turns are not full and the wheels cause the robot to have a hard time moving in a straight line as it will lean left or right. This is resolved later by the planner as it will self correct.

Navigation Program *cont.*

- Waypoint Navigation Program
  - The fetch robot comes with an already existing way to set a move goal and a planner that will help it get there.
    - Issuing the command to go to the goal was working fine, the issue was the planner was not the best so we end up using a different planner which is in an earlier slide.
  - The program borrows code from the class MoveBaseClient() and we take from it the move base initialization and the “goto” function.
  - We also created another class that would help get the initial spot of the robot at the home location making sure it finds itself on the map as it can get lost if not done properly.
  - The program uses a dictionary with numbers 0-13 mapped to a xy tuple that represents the location on the map.
  - Once the user picks a location, a move goal with that xy tuple is sent to the robot and it will then reach the location as best as it can and then ask if you would like to go somewhere else or terminate.

Navigation Program in Action



```
fetch@freight1339:~$ rosrun beginner_practice waypoint_navigation.py
```

```
launch htt
stom.launch ht
scan
nflater"
stacles"
scan
flater"
                                                    planner/Teb
PlannerROS
[ WARN] [1755610442.591075124]: TebLocalPlannerROS() Param Warning: 'altern
_time_cost' is deprecated. It has been replaced by 'selection_alternative_t
ost'.
[ INFO] [1755610442.593359929]: Footprint model 'point' loaded for trajecto
timization.
[ INFO] [1755610442.593474673]: Parallel planning in distinctive topologies
led.
[ INFO] [1755610442.593550570]: No costmap conversion plugin specified. All
pied costmap cells are treaten as point obstacles.
[ INFO] [1755610442.655616443]: max_vel_trans is not set, setting it equal
x_vel_x: 0.4
[ INFO] [1755610442.655897678]: Footprint model 'point' loaded for trajecto
timization.
[ INFO] [1755610442.738079911]: Recovery behavior will clear layer 'obstacl
[ INFO] [1755610442.746593615]: Recovery behavior will clear layer 'obstacl
[ INFO] [1755610442.769560575]: odom received!
```

Lessons learned

- Technical difficulties are more frequent in robotics than we originally thought.
- We learned how to debug larger systems, which was more complex than the systems we have debugged in the past.
- In addition to learning how to work in ROS, we also learned how to work with the ROS Navstack.

Future Goals

- We would have liked to add voice recognition to the program to replace typing in the terminal, however this would involve updating python from 3.6 to 3.9, which we were not prepared to do.
  - In the future, we would like to update the python version on the robot and implement the voice recognition feature.
- Increasing the speed of the robot or having a way for users to adjust speed to their liking would also be a future addition.