**Clutterbot Assessment**

**Simulation Setup:**

1. I have used Turtlebot3 with ROS2 foxy.
2. Made 2 configurations for changing angular resolution (0.9,2.5)
3. Bonus part has been done and added in outputs folder.

**Part1 : SLAM Comparison**

**Comparison Between Gmapping and Cartographer:**

1. **Reason for choosing Gmapping and Cartographer**To compare a Particle filter Vs Graphs based SLAM. Gmapping uses a particle filter-based approach, Each particle represents a possible pose of the robot, and the algorithm maintains a set of hypotheses about the robot's position and map.  
     
   **Advantages:** Easier to set up and configure, suitable for smaller environments, and requires less computational power.

**Drawbacks:** Prone to drift over time, less effective in larger map in my analysis

1. I tested Gmapping and Carto for 2 maps; 1 small map in which Gmapping performed fairly well and carto as well, but for bigger map Carto was much better.
2. **Loop Closures:** Gmapping has limited loop closure capabilities. It can handle small loop closures but struggles with larger, more complex loops, for the bigger map, it was a struggle to complete gmapping map.There are ways to observe loop closures even in gazebo, such as when there is some drift and you return to a place with more features, it stitches the parts which got drifted with the help of older loops.

**Real-Time Monitoring for LC in Gmapping:** Gmapping provides real-time map updates in RViz, but it lacks an explicit confidence score for loop closures, it gives a localisation measure on /entropy. And there are other ways such as ESS score but no other metric given by gmapping.

**Real-Time Monitoring:** Cartographer offers real-time map updates on terminal itself and includes a confidence score for loop closures. This score indicates the algorithm's certainty about the detected loop closures, The confidence score helps in assessing the reliability of the map and identifying areas that may need further exploration on the terminal itself

1. **Spread of particles** in Rviz is another way to visualise if gmapping is doing bad since it spawns particles in various places on the map to compare for AMCL. It gives a decent estimate of how much you can ‘trust’ localisation. It can also help us identify areas of more uncertainty in the map.  
   Cartographer does not use particles for visualization. Instead, it visualizes the pose graph with nodes and edges representing poses and spatial constraints. We can however monitor the pose graph to understand the robot's trajectory and loop closure detections.
2. I observed it is better it we do frequent rotations during mapping to capture the environment comprehensively. Straight-line movements can lead to incomplete or inaccurate maps.. For gmapping it was even more important, I think because when it drifts overtime so for larger, for an environment with less features we require to rotate more.
3. **Scale Limitations:** Gmapping can handle moderately large environments but does bad for bigger maps. I have put my results in the outputs folder, Gmapping after some distance does bad because of odometry issues, and drifts, which makes it not very useful for large maps, unless we do a multi-session mapping with gmapping.
4. **Error Handling:** Gmapping can struggle to recover from significant localization errors or large mapping inaccuracies.

Example: If the robot encounters a major localization error, Gmapping might not correct it effectively, leading to persistent inaccuracies (happened a lot, and can be seen in outputs)

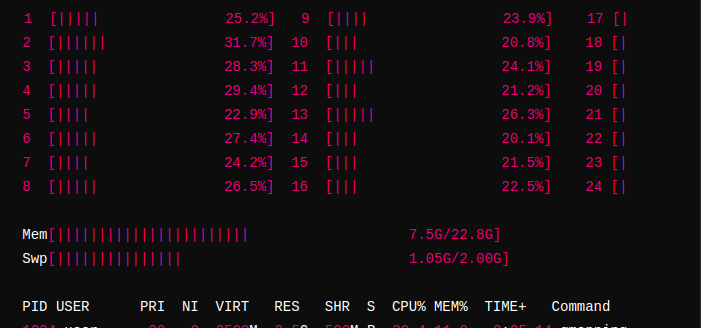
1. **Angular Resolution:**

Angular resolution refers to the smallest angle over which the sensor can distinguish between two points. It's a measure of the detail that the sensor can capture. Lower angular resolution (smaller angle) means higher detail, while higher angular resolution (larger angle) means less detail. Attached computation use for both in output.

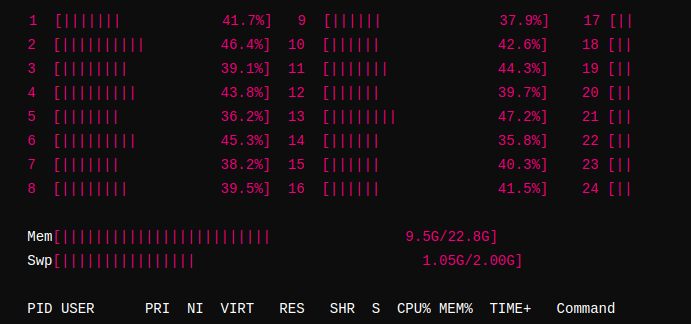
**0.9 Degrees:** This lower angular resolution provides higher detail, capturing more information about the environment, which can lead to more accurate maps but requires more computational resources.

**2.5 Degrees:** This higher angular resolution provides less detail, capturing fewer data points about the environment, which can be faster to process but gave bad maps, especially for gmapping and I couldnt complete mapping.

1. **Computational requirements:**



**Gmapping**



**Cartographer 2D**

**Gmapping:**

* **Average CPU Utilization:** 20-30% per core
* **Peak CPU Utilization:** 40-50% per core
* **Overall CPU Usage:** Approximately 30% of total CPU capacity
* **Description:** In medium complexity areas, Gmapping detects and processes more features than in low complexity areas but less than in high complexity areas, leading to moderate CPU utilization.

**Cartographer:**

* **Average CPU Utilization:** 30-40% per core
* **Peak CPU Utilization:** 50-60% per core
* **Overall CPU Usage:** Approximately 45% of total CPU capacity
* **Description:** In medium complexity areas, Cartographer’s advanced processing algorithms handle a significant number of features, resulting in higher CPU usage than Gmapping.

### **Interpretation:**

* **Gmapping:** Shows moderate CPU usage in medium complexity areas, balancing efficiency and performance. It may occasionally spike when encountering denser features or performing map updates.
* **Cartographer:** Utilizes more CPU even in medium complexity areas ensuring, and is more accurate SLAM processing at the cost of higher CPU usage.

**Part 2 : Lidar Odometry**

The rf2o (Range Flow-based Odometry) package is designed to estimate the robot's odometry by comparing consecutive 2D LIDAR scans. It uses range flow concept to compute the relative motion between scans.

The objective is to minimize the error between the observed range flow and the predicted range flow using the motion model. This leads to least squares problem and error function is defined as the difference between the range measurements of corresponding points in consecutive scans

The parameters I played around with;

**Max\_iters:** Maximum number of iterations for the optimization process, Increasing the number of iterations lead to more accurate localisation but at the cost of higher computational load. Decreasing it speeds up processing but robot localisation jumps more. (~15)

**Conv\_threshold:** Convergence threshold; Lower value leads to stricter convergence criteria, improves accuracy but requiring more iterations, so need to move robot slower during mapping (0.01)

**min\_valid\_points:**Minimum number of valid points required for a reliable odometry update (~90).I think in sparse environments, reducing this value to allow for odometry converge, otherwise it won’t be able to converge with fewer points but at the same time we will be reliant on other sources of odometry then. In dense environments, it can be increased to have faster mapping.

**A few edge cases I could think of are:**

1. **Featureless Environments:**

Lack of distinct features can make it difficult for the odometry algorithm to accurately determine movement.

I have tried implemented this. Here we take IMU data to improve pose estimation in large open spaces. It subscribes to IMU and odometry data, detecting when more than 50% of lidar points are beyond a 4-meter threshold to identify open spaces. When such spaces are detected, it integrates the IMU's orientation with the current pose obtained from the odometry. The updated pose is then published **(src/rf2o\_odom/src/rf2o\_laser\_odometry/src/large\_openspace.cpp)**

1. **Dynamic Obstacles:**

Moving objects can introduce noise and cause inaccurate pose estimation, also occlusion can prevent from it from localising well.

1. **Network Latency or Synchronization Issues:**

So lidar, IMU, and odometry data are processed by different nodes, network latency and synchronization issues can arise. This can cause misalignment in the time-stamped data, leading to inaccurate pose estimations.

Like when the robot is moving quickly, a delay in receiving IMU data due to network latency might result in the lidar odometry node processing outdated information or it will not use it and waste memory as well.

**Solution:** Implement time synchronization mechanisms such as using ROS Time Synchronizer to ensure that the data from different sensors are properly aligned based on their timestamps. Or some something like DDS.

1. **Computational Overload:**

Running computationally intensive algorithms like rf2o/TEB etc on hardware with limited processing power can lead to performance bottlenecks, causing delays in odometry updates and potentially leading to degraded navigation performance.

**Solution:** Optimize the computational pipeline by offloading some of the processing to dedicated hardware accelerators (e.g., GPUs or FPGAs). Additionally, we can use lightweight algorithms and efficient data structures to minimize CPU usage. Implementing adaptive processing rates based on the robot’s speed and environmental complexity can also help manage resource constraints effectively.