SLAM Challenge Report

(ROB 330)

Alexandru Otlacan (aotlacan) and Akash Sarada (asarada)

Overview

This report details the implementation, results, and analysis for the **SLAM Challenge** tasks in **ROB 330**. It includes discussions on mapping, localization, and performance tuning, as well as reflections on the challenge.

Task 2.2 – Occupancy Mapping (15%)

Cell-Odds Response: The cell-odds representation was chosen instead of probability due to the smaller computational cost of adding logarithms versus multiplying small probabilities. This allows for the robot to compute the odds of each cell much quicker and more accurately. Furthermore, the additive step removes the worry of numerical stability, as computers cannot compute or store extremely small or large numbers. By simplifying the multiplicative step to an additive step, we can reduce this worry as we have a larger range of values to work with, instead of bounding the values between 0 and 1. Furthermore, if a probability is small enough to be stored as 0, then adding 0 does not drastically change the odds, while multiplying by 0 would set all further probabilities to 0 as well, which could cause errors.

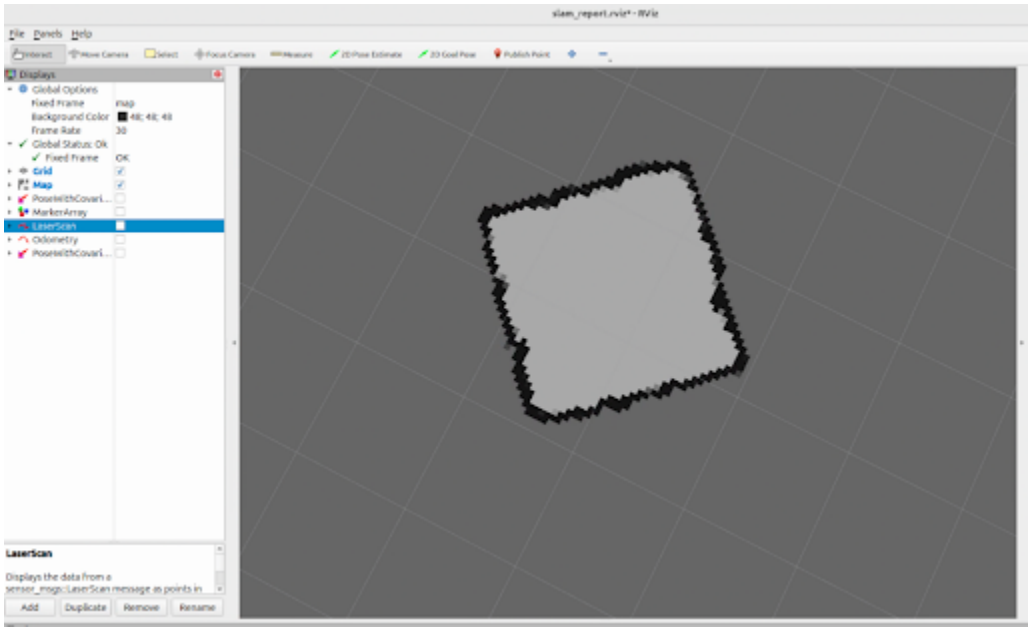

Figure 1: Occupancy map from `square_still2` bag.

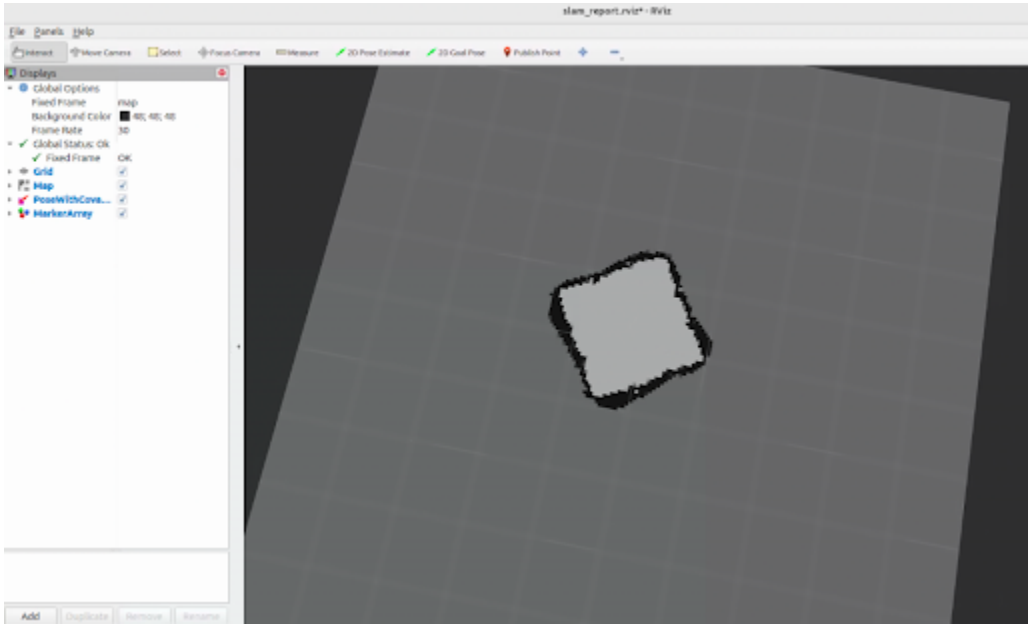

Figure 2: Occupancy map from `square_move` bag.

Alternate Values Response: Changing the values for `kHitOdds_` and `kMissOdds_` changed the way the map updated. Increasing `kHitOdds_` meant that the map found the walls faster, but would take longer to remove any artifacts from previous scans. This meant

that when it would often find walls that were not present, and would take longer to remove them. The details were crisper, but there were more mapping errors overall. However, increasing `kMissOdds_` meant that it would take longer to find the walls, and would remove any artifacts very quickly. This resulted in gaps forming in areas of the wall it had not properly scanned, thinking that the cell was free when it was not. The details were quite noisy at first, but the final map was much more accurate. Finding the right balance between the two gives the robot the correct amount of uncertainty, mapping out its environment quickly and accurately.

Task 2.3 – Action Model (10%)

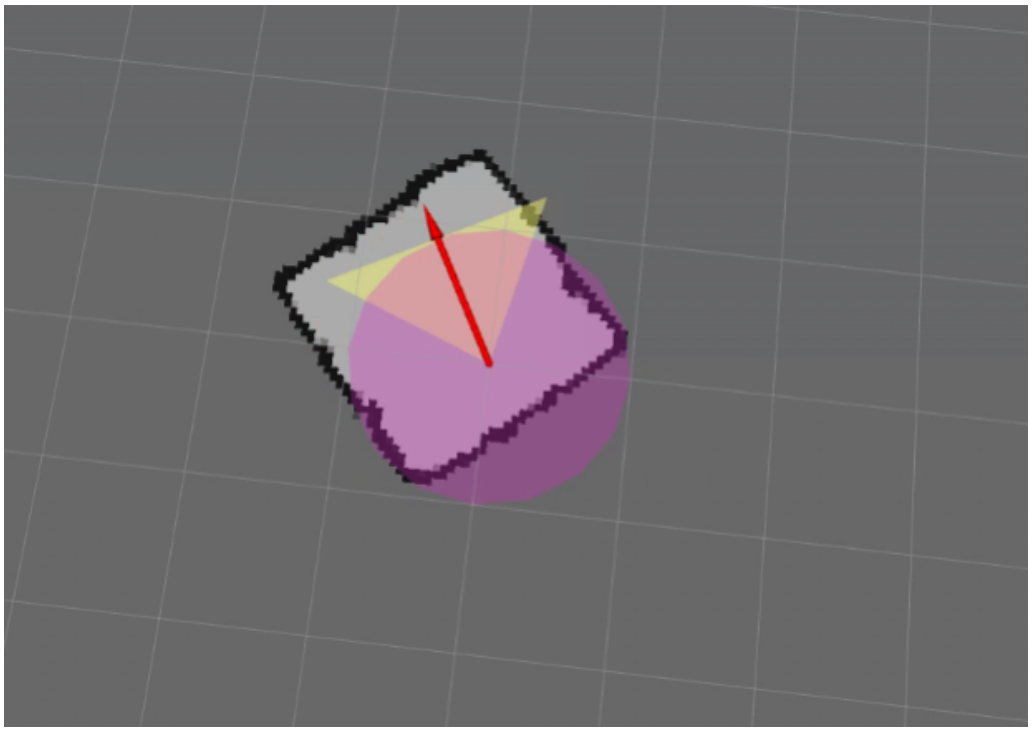

Figure 3: RViz view of `SLAM_POSE` from the action model.

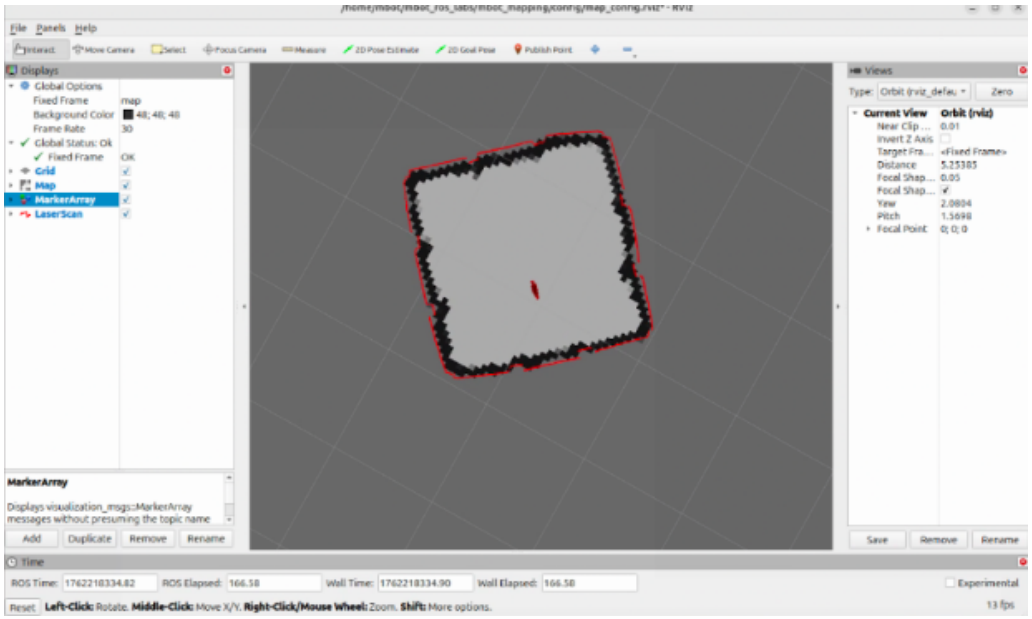

Figure 4: RViz view of `SLAM_PARTICLES` from the action model.

K1 and K2 Response: Changing the values for k1 and k2 changed the noise coefficient of the model, changing the way the robot values uncertainty. Increasing the values made the model more uncertain, increasing the blurriness and drift along the map. It also meant that the robot trusted the odometry less, which meant the robot was more responsive to the LIDAR data. However, decreasing these values made the map clearer, as it valued uncertainty less. The localization became less aggressive, trusted the odometry more, and became less responsive to new LIDAR data. However, the localization prediction started to drift as we decreased k1 and k2 more, likely due to wheel slips and other odometry errors. These values can be adjusted based on the performance of the LIDAR and odometry. Groups that trust the LIDAR more than the odometry could increase k1 and k2, while groups that trust the odometry more could decrease k1 and k2.

Particle Divergence and Mitigation Response: The divergence of particles represents the uncertainty of the robot as it continued to drive around the space. This happens as the robot continues to move, slowly losing confidence in its sensors due to accumulating errors. The way that we combated this was by implementing a resampling procedure. Resampling helps reduce the divergence of particles by choosing particles from the previous update based on their weights, sampling with replacement. Therefore, particles with higher confidence were chosen more often than those with lower confidence to be included in the next posterior.

Task 2.4 – Sensor Model (5%)

Sensor Model Response: We did not implement the sensor model differently from the code skeleton, as the skeleton allowed for an easier coding and debugging experience. However, we could improve the sensor model by implementing the beam model. This would help the robot better interpret the LIDAR sensor values, as well as better account for the modes of failure of the sensor.

Task 2.5 – Particle Filter (25%)

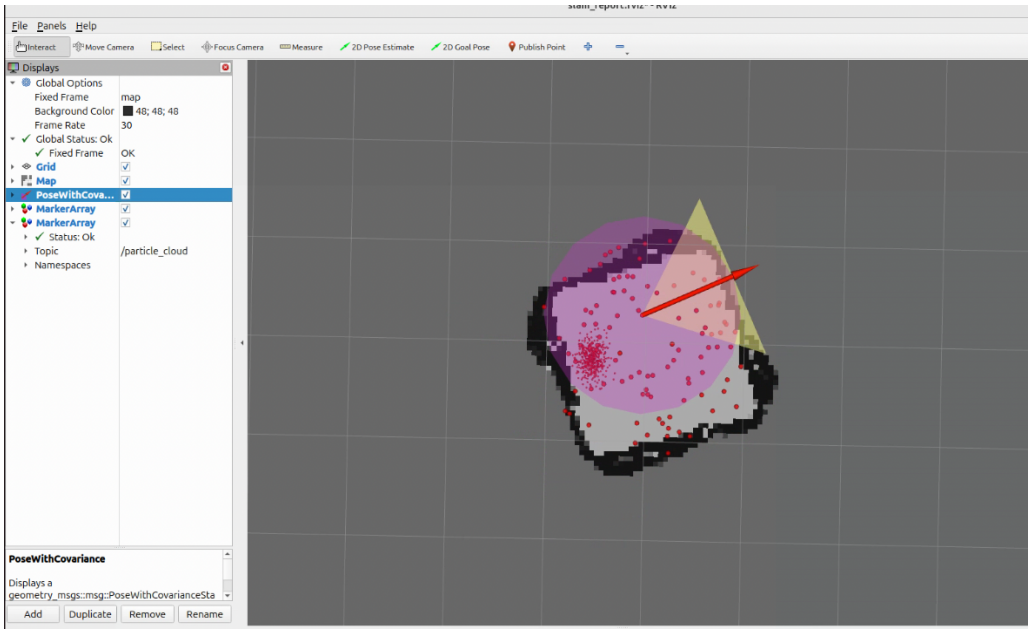

Figure 5: RViz view of `SLAM_POSE` and `SLAM_PARTICLES`.

Observations: Now that the sensor model is incorporated, the particles are centered more accurately around the robot, as both the odometry and LIDAR values are being used to estimate the position of the robot.

Tweaks: Some tweaks we had to make was updating the storage of the rotation values of the robot, which allowed for the generation of a better map, and more consistent particle filter. We started by not updating the map rotation to match the robot rotation. This led to a very noisey representation of the square. However, by updating the map rotation with the robot rotation, we were able to get a more stable map, allowing for both the mapping and localization to be more accurate.

Table 1: Particle filter update time per cycle

Particles	**Time (s)**
100	0.13
300	0.13
500	0.13
1000	0.13

Impact of Particle Count, Bottlenecks, and Tradeoffs: The impact of increasing the number of particles is the computational cost, specifically the time it takes, for each update to occur. This is could also the bottleneck, as the robot has a limited amount of computational power, and thus cannot compute too many particles at once. Adding more particles allows for the localization to be more accurate, but adding too many particles means that the robot slows down significantly, creating a delay between movements. However, during our testing, moving from 100 to 1000 particles did not show any significant reduction in performance, meaning that the bottleneck does not lie till extremely large jumps in the number of particles.

Throughput Estimate at 6 Hz on the RPI: We estimate that the number of particles our filter could support running at 6Hz could be well into the millions - about 10 million points. This is because the RPI's that the mbot uses are very advanced and can perform many trillions of calculations per second.

Bayes Filter Explanation: The components of a particle filter we implement represents a Bayes' filter for robot localization by creating our updated particle filter through the use of a prior belief state, a prediction based on our odometry, an update based on the LIDAR values, and a normalization to ensure the probabilities are in the correct range.

Task 2.6 – SLAM (15%)

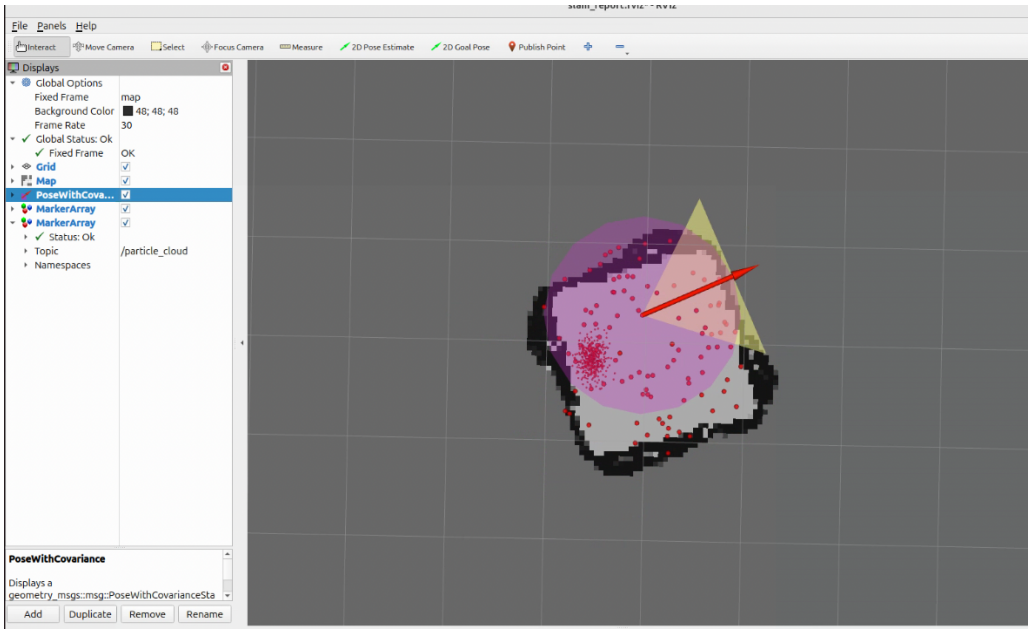

Figure 6: RViz view of `SLAM_MAP`, `SLAM_POSE` and `SLAM_PARTICLES`.

Data Comparison Response: Qualitatively, the raw odometry node updated much faster than the implementation node, allowing for the odometry to be more precise about the differences in poses between states. However, as the robot moved, it accumulated drift, meaning that the odometry become less accurate over time. On the other hand, the SLAM node stayed accurate as it was able to use the LIDAR values to correct for the drift present in the odometry node. Quantitatively, the odom node saw a difference of 0.333 meters in the x direction and -1.643 meters in the Y direction, resulting in a distance of 1.64 meters traveled. However, the slam node saw a difference of 0.452 meters in the x direction and -1.569 meters in the Y direction, resulting in a distance of 1.62 meters traveled. This shows the differences between the nodes, as the odometry node showed some of the wheel slippage, which the SLAM node was able to correct for.

Differences Response: Some possible sources of differences between the raw odometry and our Implementation could come from the slipping of our wheels. Any slippage would report a larger values for the odometry, something that would not be reflected in the overall SLAM implementation, as it would use the LIDAR values to correct its position. Further differences could have come from the update rates for the odometry and full SLAM implementation, as the odometry node is able to update much quicker than the SLAM node, allowing for more data points to be collected.

Reflection (10%)

Reflection Response: Overall, our performance was lackluster, as we did not spend enough time ahead of the challenge debugging and testing our solution. On the day of the challenge, we ran into a catastrophic ROS error, which forced us to reflash both robots, taking up the entirety of our lab time. Therefore, we were only able to get the mapping node working before the end of the challenge. Although we both had very busy schedules over Fall break, if we were to redo this challenge, we would have spent more time working on debugging and testing our code, which would have resulted in a much better score and understanding of SLAM as a whole.