Team 08: Behrens, Elwell, & Hage

Lab 04

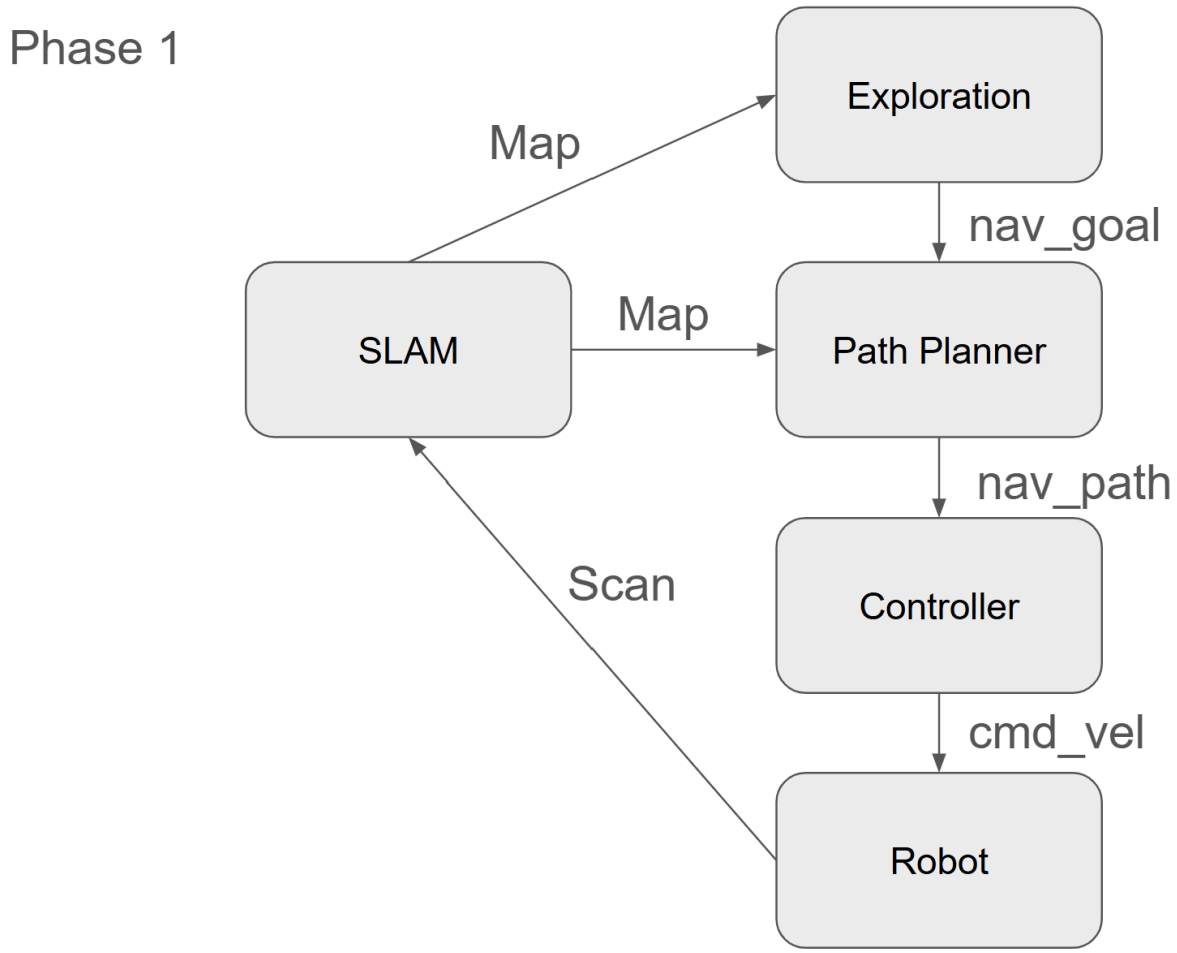
Links to releases:

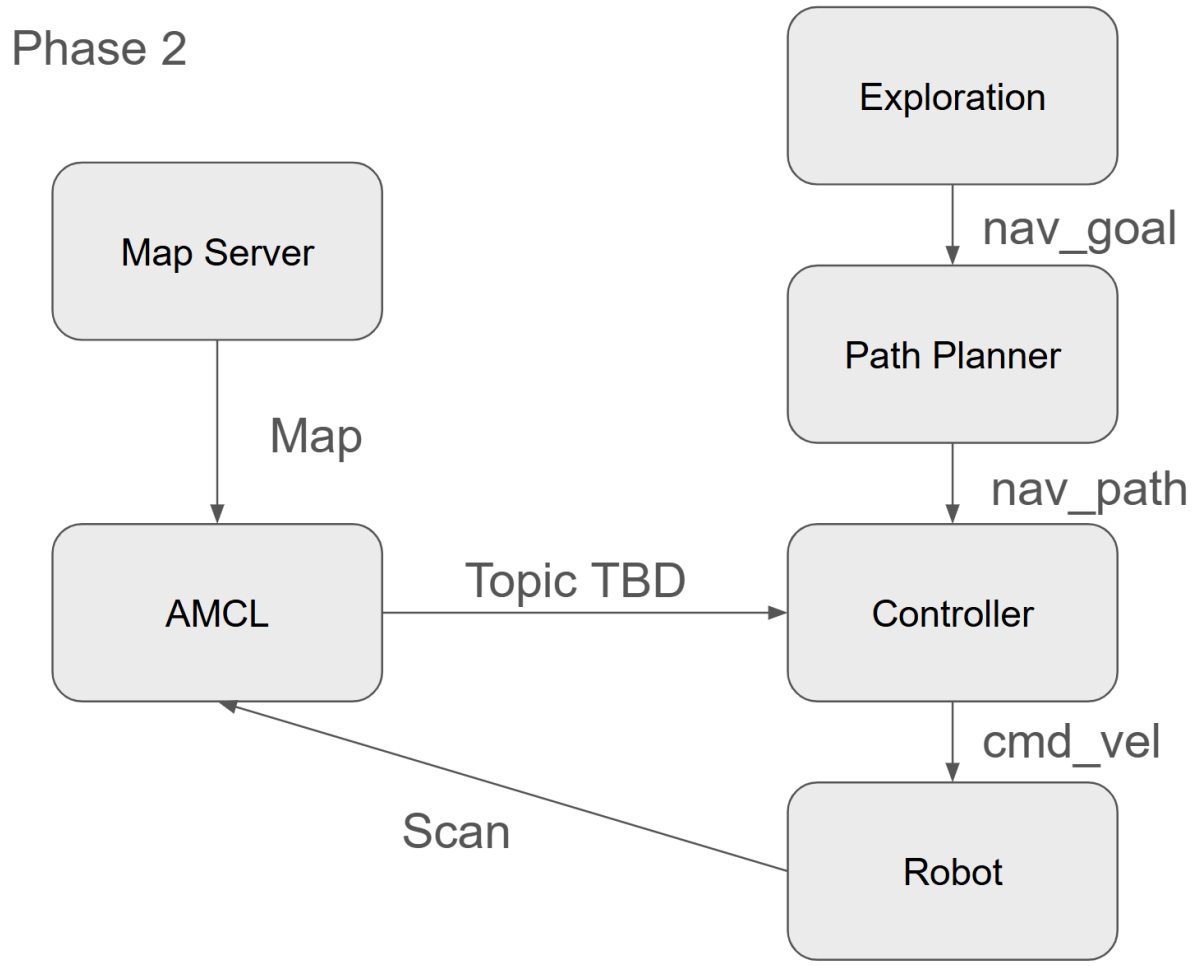
- lab4-team08

The introduction effectively presents the objectives and purpose of the lab

The robot must have the ability to search in an unknown area and start creating a map based on the new sensor readings from its LiDAR sensor. Once this map is completed, the robot should be able to store a map file for later reference, and using probability arithmetics, the robot should be able to estimate where it is within the field.

Methodology gives enough details to allow for replication of procedure. How does it compare to the original plan? System diagram





https://docs.google.com/presentation/d/1FAxEEMaw-q1koEJZklbK30GxkaujXOaxpmaNxoYDKDs/edit?usp=sharing

The overall structure of the nodes and topics of our program remains very close to what we estimated it to be from the beginning.

Phase 1, SLAM:

Initially, the robot starts in one of the field's corners, and it starts scanning. Once a sensor reading (a new map) comes in, it localizes the “frontier” cells (the pixels that the LiDAR has labeled as unknown that border known cells) and sends the centroid cell of each frontier to the path planner. When the path planner receives this information, one of three things happen. If there is no current goal frontier, the closest to the robot is selected. If the goal frontier is still present, keep this goal. If the current goal frontier is not present, approach the frontier closest to the current goal. Once a goal is established, the A* algorithm calculates its path (using cells that are walkable and away from dangerous proximity to obstacles (c_space)). With the path

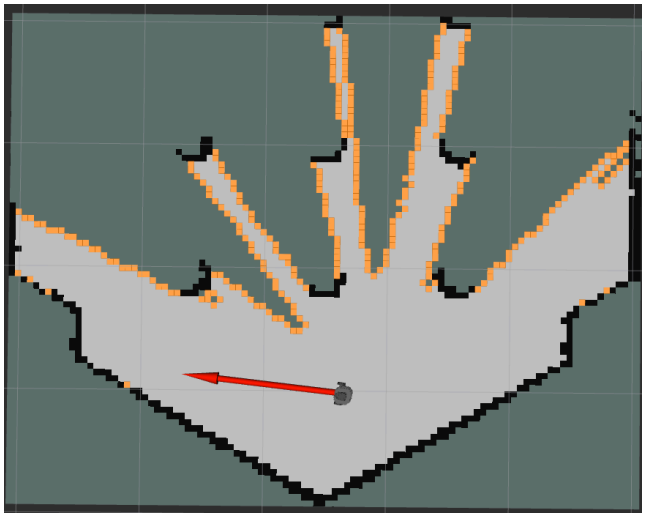
completed, the controller navigates the robot to the path's sequence of poses, taking it to its desired destination. Once the map has been completed (no more frontiers are found), the robot saves the map it created and drives back to its point of origin.

Phase 2, AMCL:

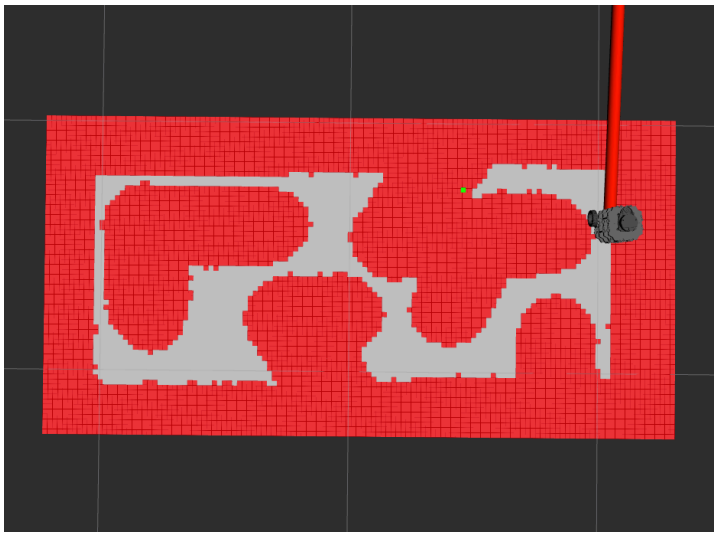
Once the map from the first phase is completed and saved, the robot can be placed anywhere in the field. When the robot is activated again, it spins a given amount of time in one direction, followed by another given time to rotate in the opposite direction. While this happens, the AMCL runs the math for the possibility of the robot's location. Once the robot finishes spinning and the probability particles consolidate to the robot's actual location, we send a location for the robot to approach within the field.

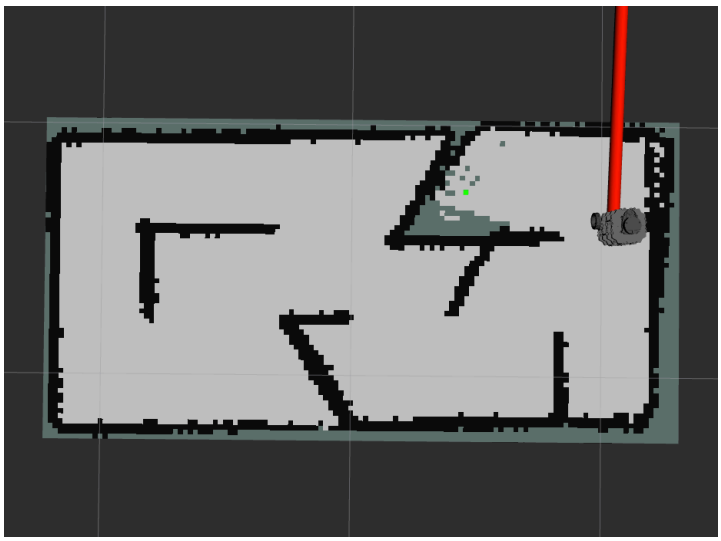
Results open with an effective statement of overall findings, present visuals clearly and accurately, present findings clearly and with sufficient support

The frontier detection algorithm has several steps. First, we find all unknown cells that are next to known cells. Then we remove all cells that are unwalkable. Next, we cluster the frontier cells and remove any clusters that are too small. Finally, we compute the centroids of each cluster. Below you will find what the robot determined as border cells for the first part of the algorithm.



Below are images showing how the map looks with the cspace (the red shadings) and without the cspace (no red shading), showing the areas we consider not safe for the robot's center to be, since they risk collision with obstacles (walls in this case).





Discussion opens with an effective statement on the goals of the lab, backs up the statement with reference to appropriate findings, provides a sufficient and logical explanation for the statement, and addresses other issues pertinent to the lab.

This lab introduced us to the generation of maps using LiDAR sensors, using the LiDAR's map to also define a c-space for safe navigation, and using the AMCL program, showing us the use of probability in determining the robot's location within a provided map.

At one point, we started having time mismatches, causing errors in the code. To solve this, we changed how the program functioned to reduce dependence on old data by transforming it as new data comes in. We also became more conscious about the performance of our algorithms to ensure they don't take too long to compute data.

One thing we did notice was that if the c-space was too big, the robot would end up sealing spots to explore and hence assume there was nothing else to be done; hence, we came up with

rounding the c-space's corners to re-enable these gaps. Also, through the process, we noticed that the robot would sometimes take too long computing the path since there were too many cells due to a high resolution. As a repercussion it would not respond in time if its current trajectory would lead it into a wall As a result, we tried adjusting to a lower resolution and hence a smaller c-space as well.

Another thing we ended up seeing in the robot's behavior was that the shortest path would run very close to the walls, so any error in sensor readings could easily cause the robot to crash on its current trajectory. To fix this, we first attempted to incorporate a Gaussian blur into the c-space; it worked well from time to time. Given its unreliability, we came up with another approach. We have two c-spaces, a primary, more strict one to push the robot away from the walls as much as possible, and a secondary, a more forgiving one, in case the primary would seal the space, as explained previously. This approach solved another issue we were having, where the farthest edges of where the LIDAR could see had much more noise than close up. So the robot would assume paths were not reachable at a far distance with the larger c-space. Using a conservative c-space for sections of the path in proximity to the robot and a more lenient c-space far away, the robot would have time to update its trajectories to be safe as it approached, while not discounting a frontier until it was close enough to have a good reading of it.

In regards to the cell selection, we already mentioned how the cells are found and how we divide them into segments to find their respective centroids. Still, we had the scenario that, as the map updated, the cells of the frontiers would be selected, and given this, the robot initially selected the cell based on the robot's proximity, leading it to change course from a neighboring point and eventually have to go back to it later on. Hence, we switched to initially using the robot's distance as a comparison, but once a goal cell has been made the first time, we determine all next cells to approach based on their proximity to the previous goal cell.

The next step upon finishing the map was to return the robot to the map origin. The only case that we had to use for the map being finished was if there were no remaining frontiers to discover, suggesting that all borders in the existing map were solid walls. In the case that the mapper yielded zero frontiers, it would send a list consisting of the map origin to the path planner, as opposed to a list of frontiers to discover. In theory, this idea was simple and should be easy to implement. However, because the robot often would finish mapping at the opposite end of the map, the A* Path planning back to the origin tended to be computationally intense. As a result, the robot would wait in a neverending deadlock between the mapper and path planner nodes.

In the demonstration, we were not able to solve this issue, but believe that it can be solved by shutting down the mapper node upon saving the map. The issue was that the mapper node would repeatedly send frontiers (in this case, simply just the origin) to the path planner. Each time the path planner received frontiers it would plan a path to the nearest one. Upon finishing planning this path, it would send the path to the controller, but not in enough time before the mapper sent another path from the robot's current position to the origin, resulting in another path

starting from the robot's position, never completing it. If the mapper node were to be shut down upon saving the map, no more frontiers would be sent, resulting in the robot only receiving one path to the origin for it to follow.

The next step of the process was using AMCL to solve the kidnapped robot problem. At first, the AMCL node was difficult to work with simply because not much documentation of it exists for ROS2. We were able to initiate the AMCL node, but we hit a wall when attempting to visualize the particle cloud and pose with covariance, and the node kept asking for an initial pose. To combat this, we simply had to call the service "/reinitialize_global_localization", which would wipe any preconceived notions of where the robot was in the map from the AMCL node's perspective. As a result, the node would send out a particle cloud of poses being tested and a "best fit" pose with covariance. We simply added this terminal command as an executable process at the end of our launch file.

The next step was recalculating and adjusting the particle cloud as new data came in. By calling the service "/request_nomotion_update", the node would publish its new calculation, but would only do so if the robot was moving to avoid incorrect assumptions gaining probabilistic weight. In our localizer node, we added a publisher to "/nav_path" containing a Twist to move the robot as AMCL was localizing the robot. To avoid the robot crashing into walls, we simply gave the twist a rotation about the z-axis. After tuning AMCL values, the robot's position in the map would converge in mere seconds.

Now that the robot's pose is recognized, we subscribed to "/clicked_point" to give the robot a new pose to path towards. Because the robot's position in the map wasn't 100% certain, we let the localization keep running to continue adjusting its position in the map, resulting in slightly clunkier driving. Nonetheless, the robot was able to navigate to its new pose with ease, completing phase 2.

The conclusion convincingly describes what has been learned in the lab

This lab taught us how to use c-spaces for navigating safely through the field, avoiding obstacles. We also learned about time-stamped messages for determining older or newer data, as well as the importance of keeping the frame consistent throughout the algorithm's methods and preventing offsets in the robot's data. Additionally, we learned how important performance in algorithms is for real-time robotics and that sacrificing sensor resolution can give significant speed-ups in other parts of the program, like path planning.

	Chase Behrens	Gavin Elwell	Charbel Saeed Hage
Coding SLAM	40%	20%	40%
Coding AMCL	20%	60%	20%
Write Up	30%	30%	40%
Total:	33%	33%	33%