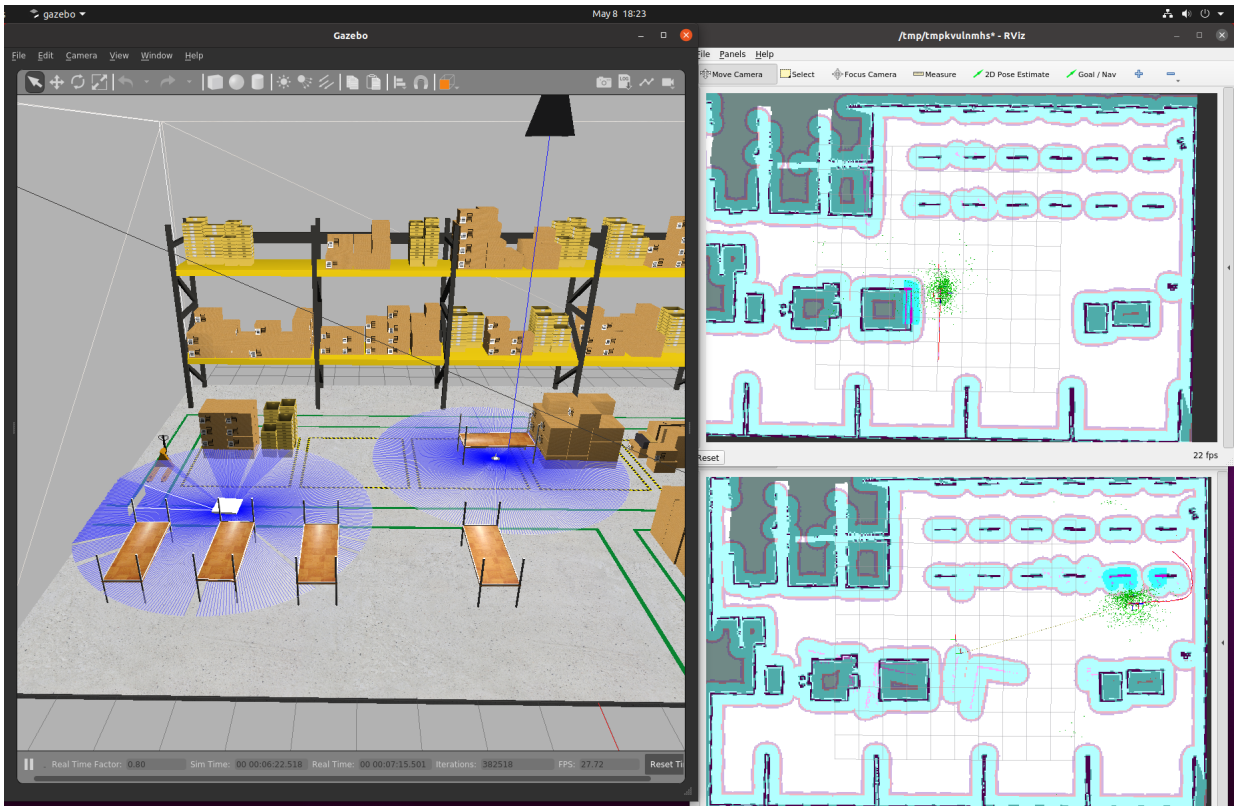
Collaborative Robotics Team #13

Robert Ronan rr3749
Ishwari Nalgirkar Ian4133
Reuben Juster rmj332

Successes

Even though our group was not successful completing all the tasks, we were able to get the amazon-warehouse-multi-robot Docker container running in both a linux virtualbox on our own computers and in AWS Workspaces. Using workspaces was necessary for the members of our groups with limited computing resources.



A Gazebo window showing the project alongside an RVIZ window for each robot.

Github Contents

Our planning file can be found at:
https://github.com/rlronan/multi-robot-exercise/blob/main/multi-robot-edit_2.py

Obstacles Encountered

We encountered many obstacles, some of which we were able to overcome and the rest we would be able to overcome with additional time and experience.

Environment

Each group member could not get the native version of amazon-warehouse-multi-robot working on their computer. This would have been ideal since docker containers are more difficult to modify and interact with.

Each team member attempted to run the Docker version directly on their computers, but there were issues with some of the dependencies for the Container to interact with the local environment.

The team members tried running the naive version on a Ubuntu VM via VirtualBox on their computer. The ROS2 Turtlesim tutorials were able to run, but they encountered similar problems to their local environment when they tried to run amazon-warehouse-multi-robot.

Each team member was able to get the Amazon program to run the docker container on their local VM, but only one team member had a powerful enough computer to make it run properly.

The other two team members were able to get it to run via docker on an Amazon Workspace VM.

Robot Pathing & Localization Issues

Localization and path planning are dependent on each other for proper navigation. A deficiency in one throws off the other.

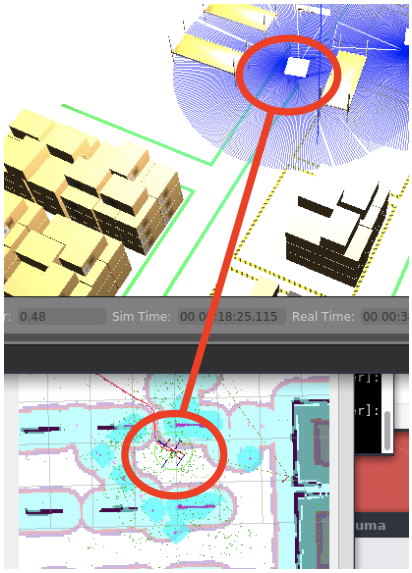
The Ros2 Nav2 navigation setup consists of both a global and local planner. The global planner is responsible for determining an approximate path consisting of waypoints from the current location to the intended destination. The local planner is responsible for managing the navigation from waypoint to waypoint. This responsibility includes navigating the robot around known obstacles in the map, as well as avoiding collisions with unknown objects that are identified via the robot’s sensors.

We identify a number of significant issues, which we discuss below, with the currently implemented navigation system that lead to the robots failing to properly perform the repeated sequence:

1. Move to a non-stored pallet.
2. Load the pallet.
3. Move the pallet to storage location.
4. Unload the pallet in the storage location.

Poor Localization

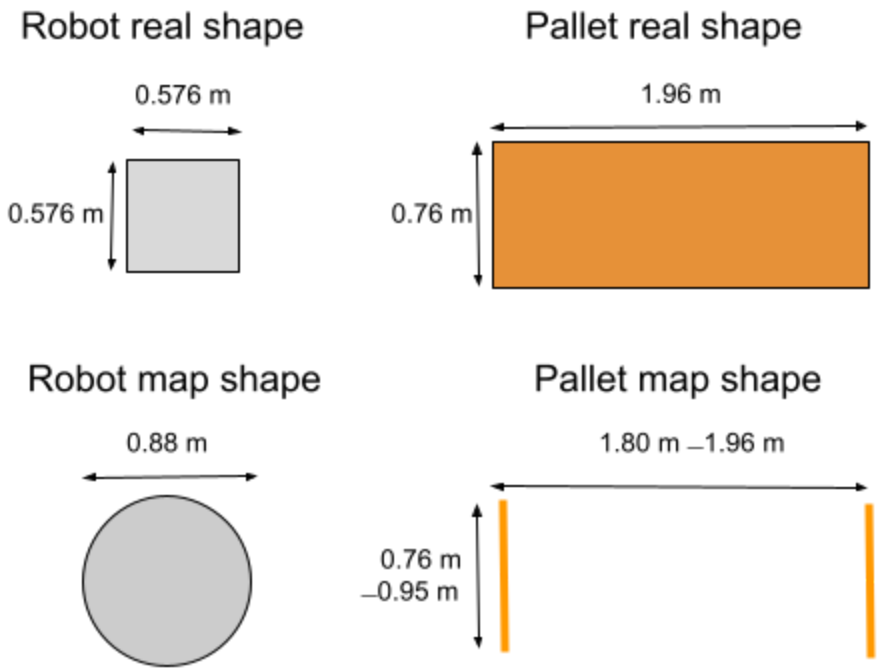
Localization is initialized by going to each robot's rviz window and using the 2D Pose Estimate tool to set the robots' belief of where it is on the map. If this tool is not used, the robot never gets localized. There is little room for error using the tool. Even if the robot is properly initialized, multiple problems, as shown in the following sections, will throw off localization.



Example of completely inaccurate localization

The navigation system is context-free

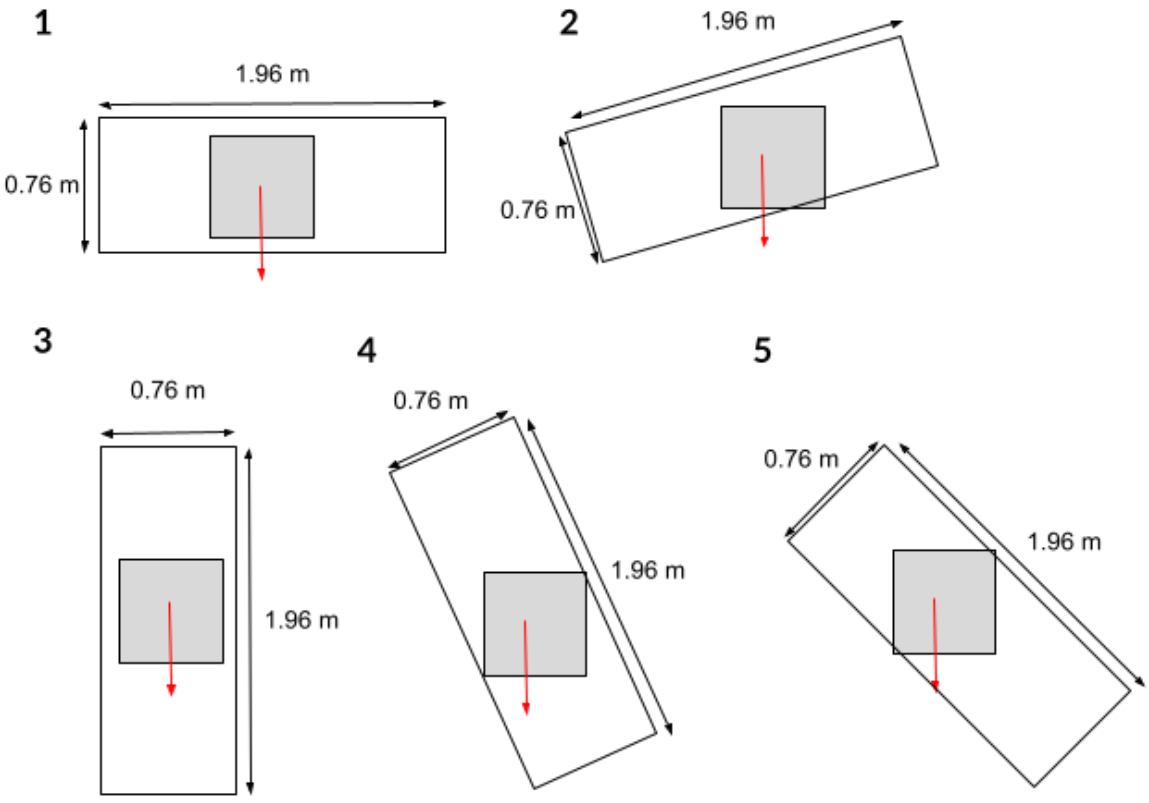
The global navigation system is unaware when the map changes and cannot generally remove objects. For instance, the robots will attempt to avoid colliding with their map's representation of the pallet they just picked up, because a robot 'loading' the pallet does not invoke a change in the robot's global map.



The real shape of the robots and the pallets, and their internal representations in the RVIZ map.

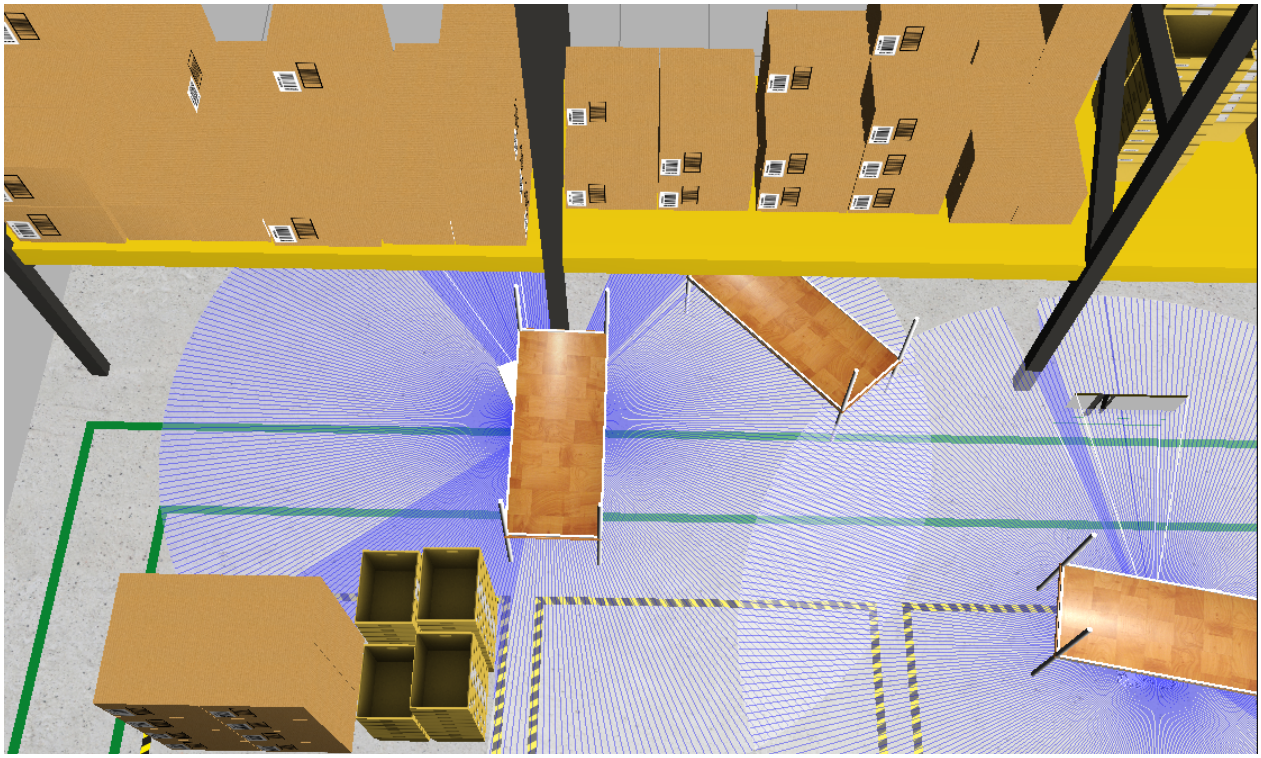
The local navigation is unaware of the robot's condition when carrying a pallet. Specifically, the navigation system has no knowledge of whether the robot is carrying a pallet or not. Nor does it have any knowledge of the relative change in shape of the robot when it is carrying a pallet. That is, the navigation system does not recognize that it must take into account the pallet's shape and position on the robot to avoid collisions. Given that collisions frequently lead to a complete localization failure followed by a complete navigation failure, we want to avoid having the pallet the robot is carrying collide with anything. Collisions could also damage both the robot and the pallet.

Thus, when the robot is carrying a pallet, representing the robot by its own shape and location is not sufficient for navigation, because the robot is carrying a large object which protrudes outward in multiple directions. Therefore we say the robot has a 'relative change in shape' when it is carrying a pallet, because in order to avoid collisions, the navigation must take into the count the shape of the robot with the added pallet.



Various positions the robots carried pallets in, due to the pallets spinning when the robot turned, or an incorrect assumption about the position of the robot.

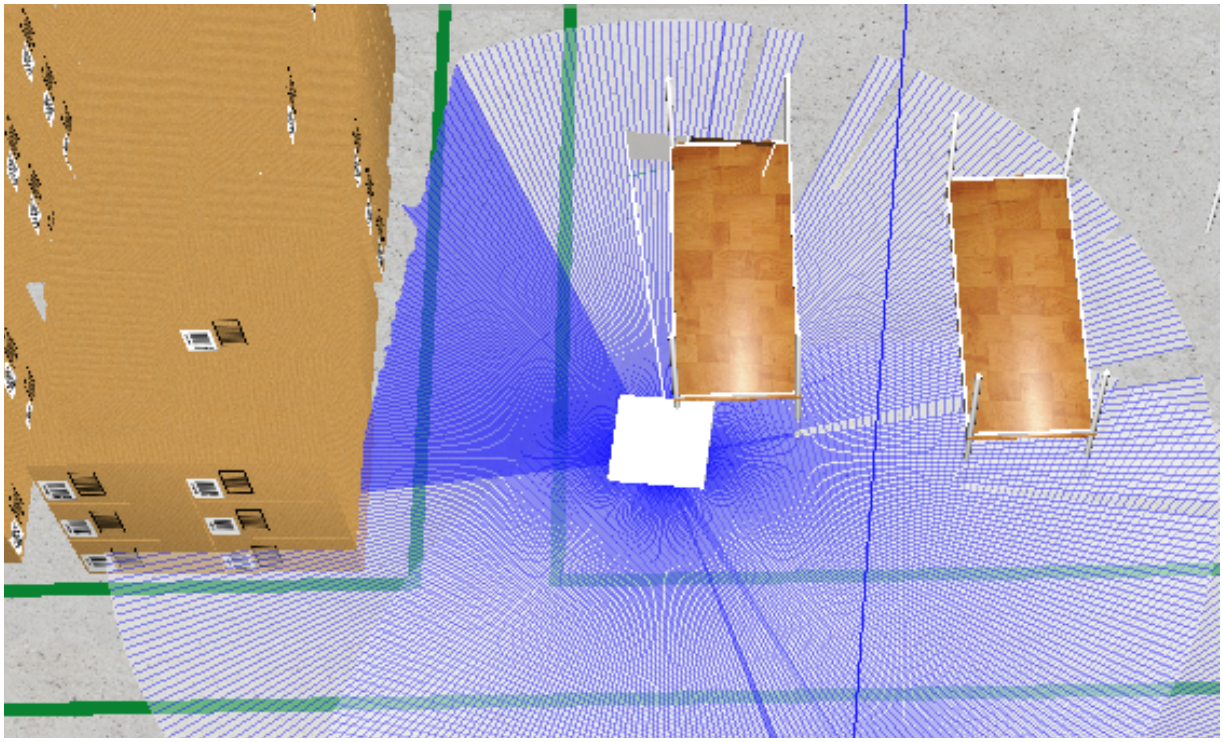
The robot often slams its pallet into objects when moving, sometimes causing the robot to get stuck or become slowed. By default, the robot never backs into or out of locations, which can cause collisions between the pallets and the storage location walls. This also eliminates several possible paths from the robot's current position and the goal pose.



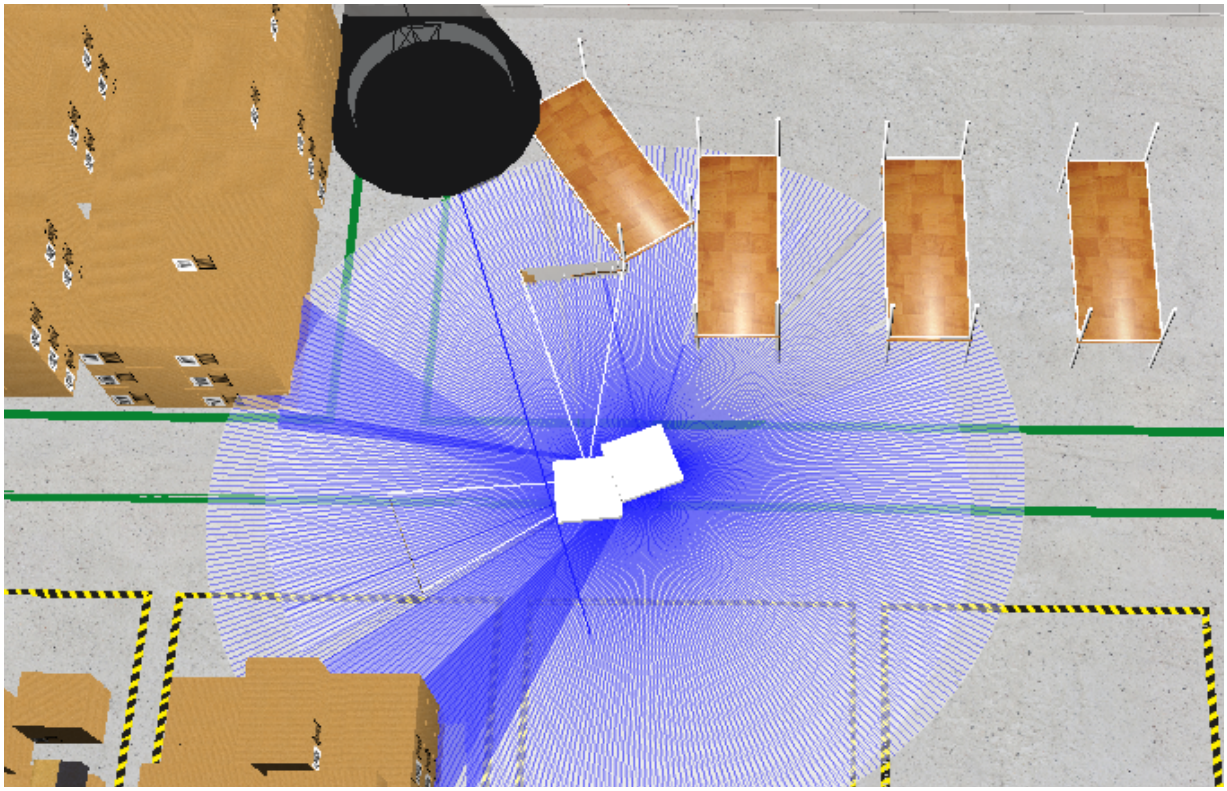
A robot runs the pallet it’s carrying into an obstacle and becomes stuck.

Local navigation fails to avoid known and unknown obstacles

The robots run directly into walls. The robots also run into each other. Each robot is unaware of the other robot’s location since they are not made aware of each other via the controller. However their lasers should enable the robots to detect and avoid each other, but in practice this is not always the case.



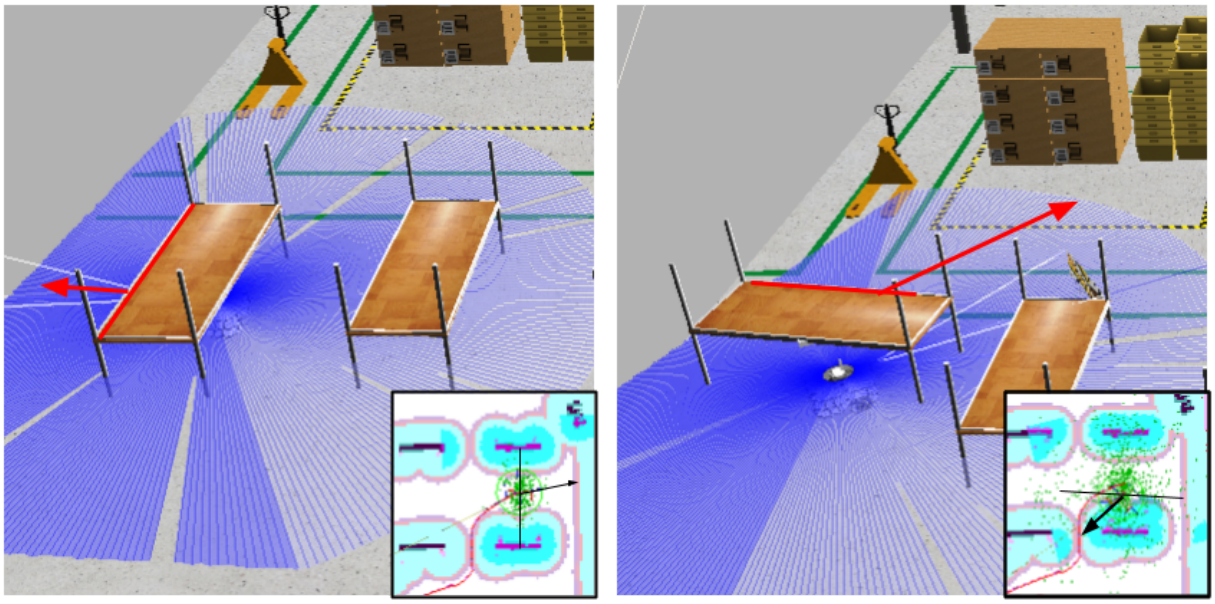
A robot begins a simulation by running directly into an obstacle.



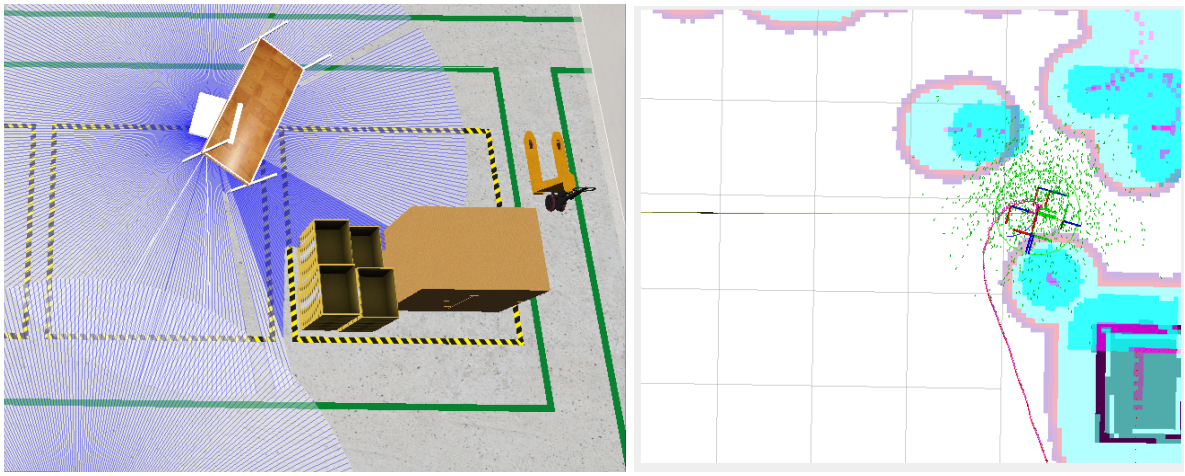
Robots crashing

The rate of linear and angular acceleration is too high
Given that the robots have ‘loaded’ the pallets by simply balancing pallets on top of themselves, they frequently accelerate too rapidly, causing the table to slide forwards or backwards which shifts the center of gravity, and occasionally causes the pallet to fall off of the robot.

Worse, the robots often spin so rapidly that the pallet, which takes the shape of a four-legged rectangular table, rotates such that one or two of the pallet's legs appear in the robot’s LIDAR sensor. Incidentally, this appears to be one of the few instances that the local navigation makes a consistent effort to avoid an object. As a result, the robot sometimes either spins in circles, or does loops while constantly attempting to navigate around an object which is attached to the robot.



(Left) The robot loads the pallet while facing approximately perpendicular (~ 100° CCW) to the pallet.
(Right) After spinning rapidly to change direction, the pallet has shifted on the robot and is now no longer perpendicular to the robot (~ 50° CCW).

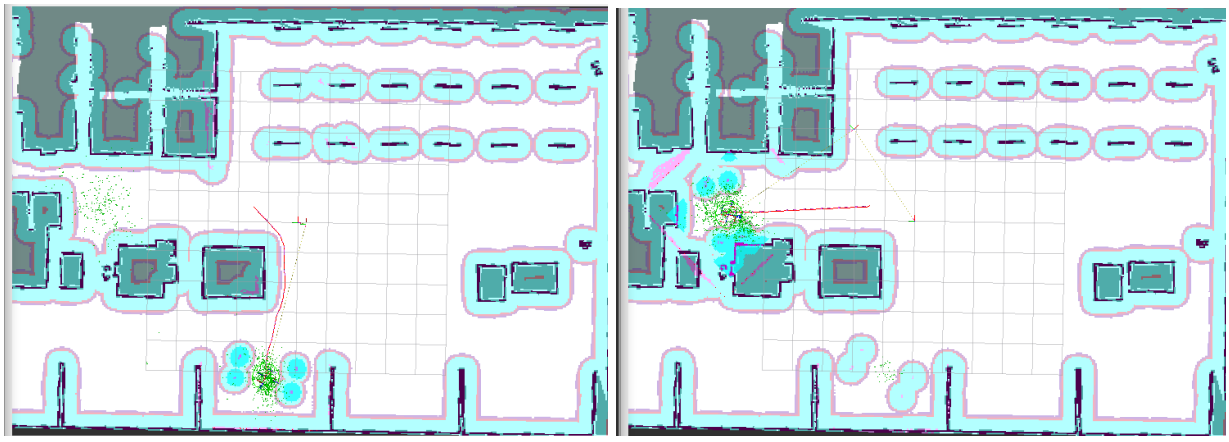


(Left) A robot dragging a pallet is attempting to navigate around one of the pallet legs, even though the pallet is effectively 'attached' to the robot.
(Right) The robot's RVIZ shows the robot's perception of a small circular object (actually the pallet leg) in front and to the left of the robot, which the robot is pathing to avoid.

Collisions cause complete localization failures, rendering recovery impossible
As mentioned, the local navigation system frequently fails to prevent the collision of the robot with objects, and more significantly it fails to prevent the collision of the loaded pallet with objects.

Since the navigation system has no inherent knowledge of when the robot is carrying a pallet, a collision of the pallet itself with an object (and then sometimes the robot with legs of the pallet) often causes a drag on the robot which is not recognized by the navigation system. In effect, the robot may be applying force to move in a direction, not registering a collision, and not moving at the velocity that force would produce. This causes a rapid, seemingly irrecoverable failure of the localization system.

The navigation system does usually attempt a relocalization based on what the robot sees with it's LIDAR. However, this typically results in either the relocalization failing completely, leading to the robot's recovery procedure failing, or the relocalization determining the robot is across the map. In the latter case, upon an inability to reconcile the navigation system's belief about the robot's location with the robot's LIDAR observations of its own surroundings, the pathing system fails repeatedly to compute a viable path, and the robot crashes.



(Left) A robot is attempting to navigate through a pallet it just placed in a storage area.
(Right) The robot's RVIZ shows the robot's relocalization attempt caused the robot to believe it is suddenly in a vastly different location.

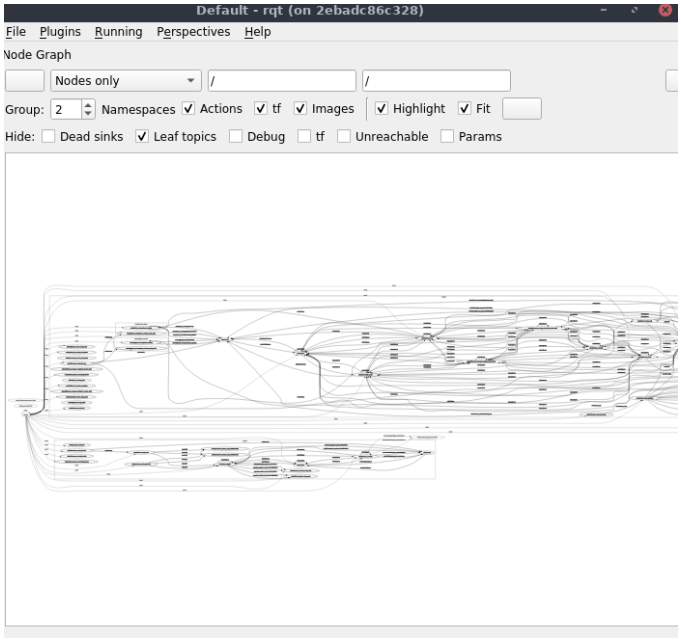
Possible solution

One of the possible solutions to poor localization and local planning is adjusting the SLAM parameters as suggested by *slam-in-ros2.* This means altering the *nav2_multirobot_params_X_with_control.yaml* file where the params are stored. The alphaXs values define the robots' noise in the X dimension. The article specifically mentioned that if the value is too low, the robot's odometry dominates the laser scans. If the odometry is poor, the robot becomes delocalized. If the alphaX is too high, the extra noise in the pose estimation causes delocalization. The article proposed some great param tuning techniques involving Rviz.

We noticed that when we increased all the alphas to 2.5 using rqt, localization and local path planning seemed to be better, except in locations without a lot of edges to detect. In open areas, the robot would constantly relocalize.

Documentation/ Source Code

Talk about issues with figuring out how to do things. What documentation we wish was out there.



RRT graph of Project

Devising a solution for the above problem involves approaches which include giving each of the robots individual isolated containers and ros2 environments. The task planner and path planner need to be modified such that the robot is able to detect moving obstacles and is able to plan a path around them without colliding. It also involves the calculation of the velocity of the obstacles and angles of the vectors to determine the modified path.

The RRT algorithm involves the nearest neighbour approach. Documentation on building the global map,the local map and obstacle avoidance, would be very helpful for implementation.

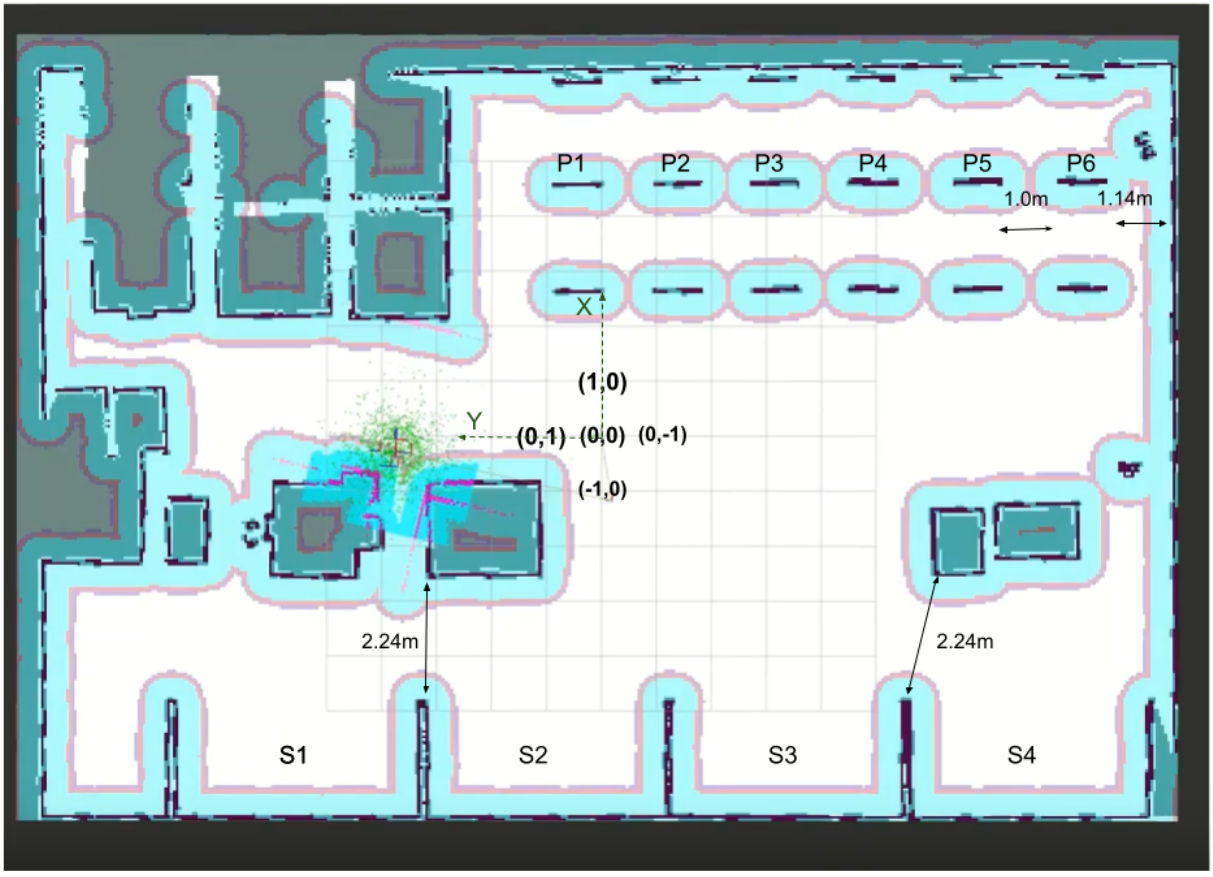
The algorithms such as A* and RRT require generation of motion profiles based on the points generated by the path planner which grows with the dimension of the map.To avoid this we could try time parameterization which involves data collection from the sensors which need to be calibrated for successful collision avoidance which is a difficult task.

To create a trajectory, velocity profile and a path. If the path is linear, the generated path requires the robot to start and stop at each waypoint to get a complete trajectory.

The programming navigation and planning algorithms problem can be solved using a Navigation stack- Navigation2 which comprises several action servers (Planner, Controller and Recovery) which are connected by a behaviour tree. The implementation involves customizing a Behaviour tree and integrating it with the behaviour palette, which is an intrinsic task.

The multi robot scenario can also be implemented by a central planner and robots with an independent navigation configuration, the task planner in this scenario is an essential part and tough to implement.

The Current Navigation System Necessitates A Specific Sequence Of Paths



RVIZ map through which the navigation system sees the world, with added labels and distances. Note that these labels differ from the original labels in the code.

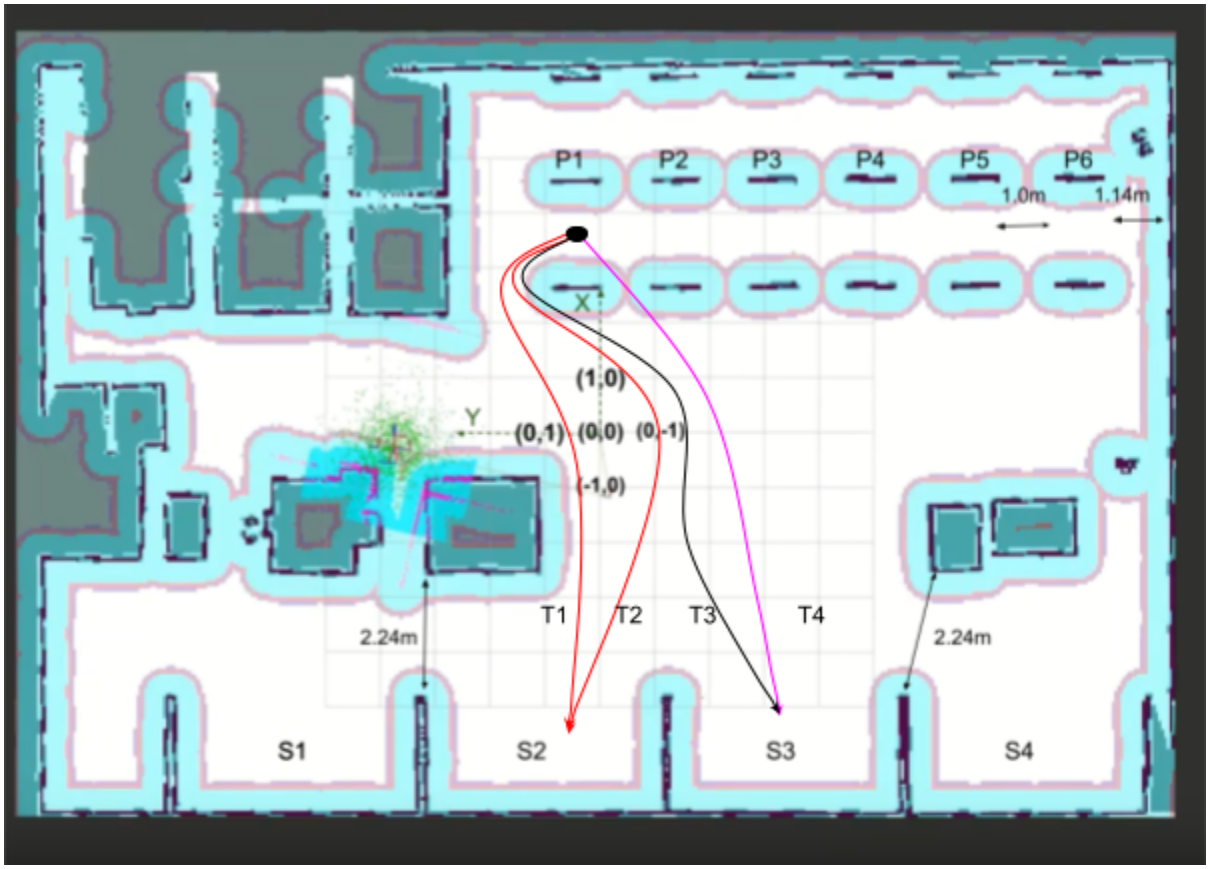
Given a small number of reasonable assumptions and what we have observed so far about the robot's navigation system, we argue that there is effectively one sequence of paths (up to equivalence by path-homotopy, which is defined below) which the robots can take to move the pallets to the storage locations.

In particular, the only viable order in which to move the pallets to storage locations is to move the pallets from left to right (P1, P2, …, P6).

Path Homotopy

By definition, two paths q1, and q2, both from point A to point B, are (path) homotopic if there is a continuous deformation from path q1 to path q2, which keeps both of the endpoints fixed. More specifically, if path q1 can be slowly deformed into path q2 without hitting any 'holes,' the two paths are homotopic. Homotopy is an equivalence relation between paths.

In our case we consider obstructions on the RVIZ map to be holes. Thus in the image below, we consider paths T1 and T2 to be homotopic (read: equivalent) to each other, but we do not consider paths T3 and T4 to homotopic to each other, as any attempt at deforming T3 into T4 would hit the horizontal line that denotes the bottom of pallet P1 (above the "X" on the map). More generally, since the storage location does not matter, we consider all four storage locations path equivalent, and so we consider paths T1 & T3, and paths T2 & T3 to path homotopic.



Paths T1 and T2 are homotopic, but paths T3 and T4 are not, as any attempt at deforming T3 into T4 would hit the horizontal line that denotes the bottom of pallet P1 (above the "X" on the map).

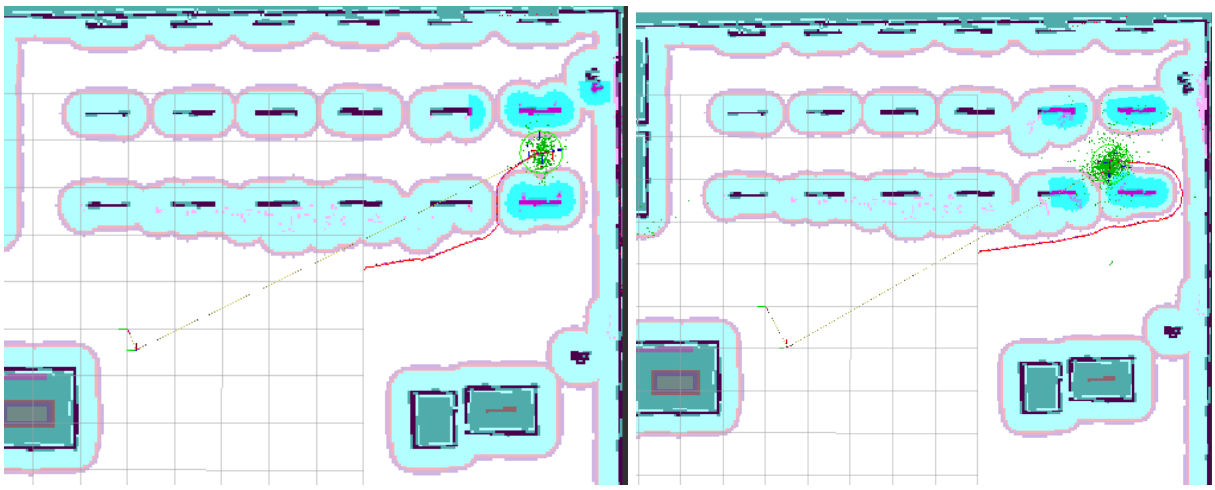
Robot Navigation Planning Assumptions

1. The robots should not drag the pallets' legs on the ground.
2. The robots should not collide with objects.
3. The pallets should not collide with objects (while the robots are moving them).
4. The robot's localization is subject to some error; in the sense that a robot's position is only known to +/- 0.2 m or so.
5. The robot should carry the pallet perpendicular to its direction of motion to reduce the probability that the pallet legs appear in the robot's vision and disorient it, and to reduce the probability of the robot dragging the pallet's legs on the ground.
6. The navigation system is context-free.
    a. The global navigation system does not take into account changes in the map (e.g. when a robot lifts a pallet, the pallet is not removed from the map, and the robot believes it is still an obstacle.
    b. The local navigation system is unaware of the robot's relative change in shape when carrying a pallet.

A Robot Cannot Viably Start With Any Pallet Except P1

Consider a situation in which a robot first tries to carry pallet P6 to a storage location. The distance between the edges of pallet P6 and the wall are shown in the map above. The pallets are 1.96 m in length and 0.76 m in width. We are assuming a robot is picking up a pallet while facing perpendicular to the long side of the pallet, as this reduces the probability that the pallet legs slide into the robot's frame of view.

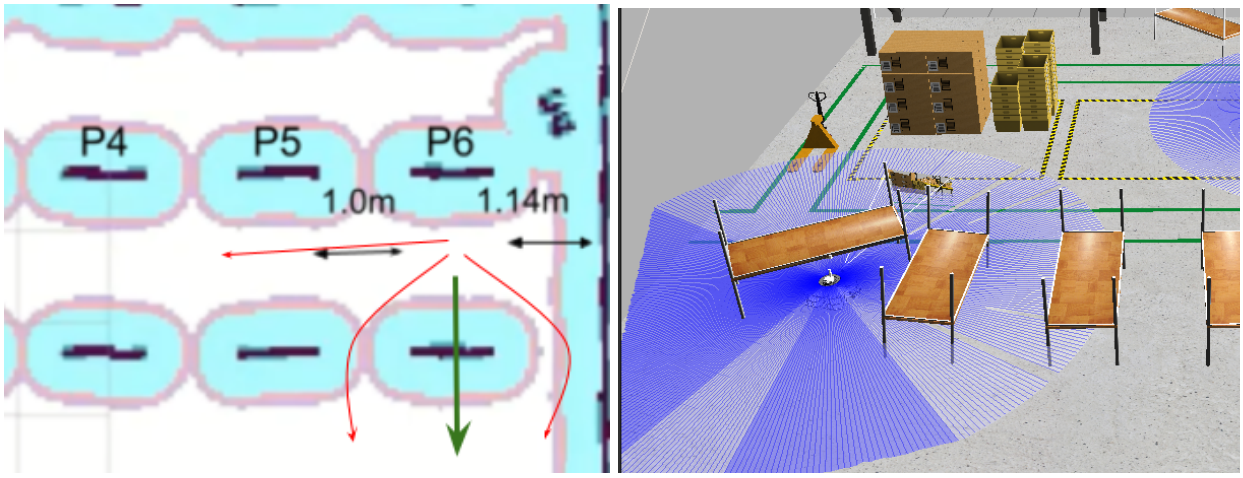
As we are assuming the navigation system is context-free, when the robot picks up pallet P6, it is not aware that the two horizontal lines (the pallet legs) which make up P6 on the map no longer exist. Even when the robot fails to detect anything in the location of the pallet legs on the map, it will still attempt to navigate around them. The navigation system is also unaware that the robot has a two meter pallet balanced on its head.



(Left) The robot is carrying pallet P6, but since the navigation system does not take that into account, the robot's path involves navigating the legs pallet P6. The area directly below the robot is empty at this point.
(Right) The navigation system makes another attempt to navigate around the bottom of pallet P6, even though the pallet is no longer there.

The global navigation will thus suggest a path that involves either moving to the left through the middle of some number of pallets, and then moving down towards the storage location, or it will suggest a path that involves moving down and around pallet P6. Given the size of the pallet, and the space available, all of these paths invariable involve a collision by either the robot, or the pallet it is carrying.

In fact, the only viable path from this P6 at this point is directly downwards, which the navigation system will never suggest.



(Left) In red, paths the navigation system may suggest, and in dark green, the only viable path.
(Right) The pallet the robot is carrying collides with a neighboring pallet as the robot attempts to navigate downward via the gap between pallets P5 and P6 on its map.

Since collisions often lead to complete navigation system failures, not to mention they are undesired in practice since it damages equipment and inventory, we argue it's thus not possible for the robots to start by moving pallet P6. Even relaxing the assumption that the robot should carry the pallet with the long side perpendicular to direction of the robot's motion, this argument still largely applies. If the robot is carrying the pallet so that the long side of the pallet is parallel to the robot's direction of motion, the robot is exceedingly likely to hit the front of the pallet on either the wall or pallet P5 as it tries to move downward and around the bottom of pallet P6.

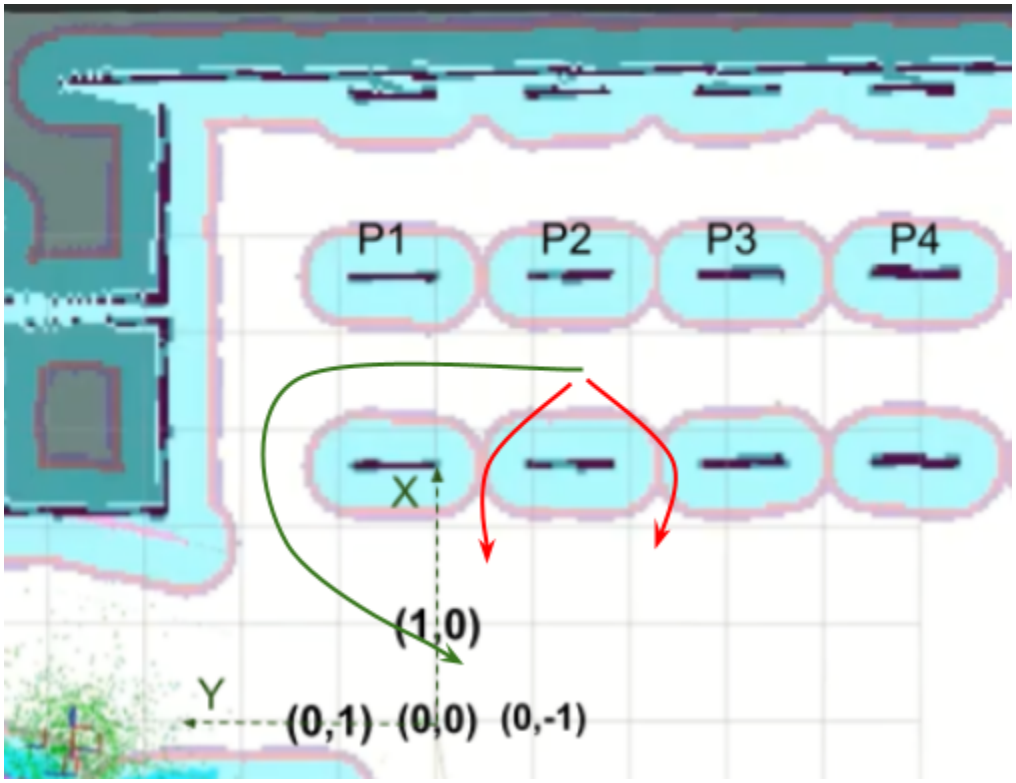
Every other pallet with the exception of pallet P1 has less space between itself and its neighbors. Therefore, it being non-viable to move pallet P6 before pallet P5 is moved also implies inductively downward to pallets P5, P4, P3, and P2. So the first pallet to be moved must

be pallet P1. Pallet P1 must be moved left and downwards in order to avoid colliding with pallet P2.

Sequence Of Pallet Moves

After pallet P1 has been moved, a robot is free to move pallet P2. However, unless the robot is directed to move left through the gap in the middle of where it believes pallet P1 to be, the navigation system will typically attempt to move down and around the base of pallet P2, causing a collision. So the robot carrying pallet P2 must be ordered to first move to the left of P1, then resume its journey.

Given that pallet P1 is now gone, if the robot is carrying the pallet parallel to its direction of motion, rather than perpendicular to it, it may be able to maneuver down and to the left of pallet P2's base without hitting anything, since pallet P1 is already gone. However, this is only a ~50% chance.



In red, two likely paths suggested by the navigation system for a robot carrying pallet P2, and in dark green a viable safe path. The robot may be able to navigate the left red path, but not the right red path, so we cannot allow the navigation system to dictate a choice.

We argue analogously for pallets P3 through P6 that the only viable path is essentially to dictate that the robot carrying the pallet move leftward until it is clear of pallet P1's location, then proceed down and to the storage locations.

It is true that if the robot tried to move through the gaps between the pallet bases that are left of the pallet the robot is picking up, it should clear the gap because none of the pallets are actually there anymore. However, in order to assure that the robot picked a gap to the left of the pallet it picked up, one would essentially have to order the robot to go left with a costmap or a waypoint, which is minimally different from what we propose here.

We argue that it is essentially only viable for the robots to pick up the pallets in increasing order, and carry them along the path shown. Technically the robots could pick up the pallets in increase order and travel through the gap between a pallet and the pallet to the left of it, but that would also require ordering the robots to move to a specific waypoint.

Ideal Implementation

If we had more time and experience, we would have tried to implement some of the ideas below.

Altering the Robot Model

As identified above, when the robot is carrying a pallet, its LIDAR sensors sometimes see the pallet legs and go haywire. We would add additional LIDARs so that the robot can still have at least one unobstructed LIDAR in the direction of travel. Since the robot's localization will go haywire in large open spaces if it is relying primarily on LIDAR, LIDARs in other directions might need to be added so that it can detect at least once surface.

Based on the RVIZ representation of the LDS, it appeared to do a good job of detecting obstacles, but the robot tended to ignore it. This might be improved by increasing the Alpha parameters and adjusting other AMCL parameters.

Pallet Storage Allocation

One of the optional goals from the *challenge* was to create a planner with the "Fastest moving of all pallets from pickup to drop-off and back". We could have used a linear programming package in python such as PuLP to solve the binary integer programming optimization problem. This optimization problem can be modified to meet some of the other goals (Fastest moving of all pallets from pickup to drop-off or Lowest average time spent per pallet per task).

$$min\ D = \sum_{r=1}^{R}\sum_{o=1}^{L}\sum_{d=1}^{L} C_{od}A_{odr}$$

S.t.

$$\sum_{r}^{R}\sum_{o}^{L} A_{odr} - A_{dd_2r} = 0\ \forall\ d \in L \wedge o \neq d \neq d_2$$

Robot r must take a continuous path

$$\sum_{r}^{R}\sum_{d}^{L_s} A_{odr} = 1\ \forall\ o \in L_p$$

Every pallet must be picked up.

$$\sum_{r}^{R}\sum_{d}^{L_p} A_{odr} = 1\ \forall\ o \in L_b$$

Every robot's beginning point must have 1 path to a pallet.

$$\sum_{r}^{R}\sum_{o}^{L_s} A_{odr} = 1\ \forall\ d \in L_b$$

Every robot must end at one of the beginning points.

o is the origin of the path
d is the destination of the path
r is the robot number
R is all the robots
L is all the possible locations
L_b are the beginning locations
L_p are all the pallet locations
L_s are all the storage locations

A_{odr} is the binary decision variable that robot r goes from origin o to destination d
C_{od} is the cost/distance/time from origin to destination
Please note the od pairs can only be bp ps sp pb.

This assumes that our collision avoidance functionality would prevent collisions in real time and that collision avoidance path deviations don't add too much time. Before the robots are deployed, the optimum path would be computed with the above equation then the robots would be given their assignment. Since the robot challenge did not have that many robots, pallets, and storage locations, the number of decision variables would be minimal and therefore the program could be solved quickly. However, this wouldn't be feasible if there were too many decision variables.

Collision Avoidance System

First we would create an action server for each robot. It appears that because the initial code had all the poses uploaded at once, a server was not needed because the robots were supposed to execute the plan without any changes. The action servers would be requested by the clients upon completion of their current goal/path.

We would create a topic per robot that is the path to the target pose. The topic's publisher would be each robots' robotx/compute_path_to_pose action server and the subscriber would be the other robot's robotx/FollowTargets action server.

In *Robot_controller.cpp*, the sections which establish the behavior of robotx/FollowTargets would be modified so that when robots' action server receives the next pose, it would calculate the initial path, check it against the other robot's current path (stored via subscription of the above topic), and modify the path based off avoiding possible conflicts. This would require adding more information from the parameter blackboard to the feedback message, which is presently only passed the current waypoint.

Robot1 would get the head start when the simulation is initialized. Then the server would send the goal response to the client.

RRT extended with Dynamic Planning

Another way to approach this problem is by using the rapidly exploring random tree algorithm which finds a collision free path at the expense of optimality .RRT* extended is a more.The RRT focuses on finding the nearest neighbour as parent for the new vertex whereas RRT* looks for the best nearest neighbor, the computation cost is an improvement over the traditional RRT.

The main steps include:

1) Moving obstacles detection
2) Path replanning
3) Path obstruction
4) Establishing replanning goal location
5) Modify search tree
6) Subpath selection and execution

This algorithm can be applied to a multi-robot problem. Dynamic replanning is used to determine the course avoiding collision with random moving obstacles.

After the initial replanning, the robots follow the optimal path produced by the algorithm with the required velocity to reach the goal state, in case of collision, replanning is initiated.

This approach does not involve use of individual robot sensors instead a detection range is used to determine whether the robot's sight to the moving obstacle is hindered by a static object.To determine this, we need to use trigonometric functions with a direction vector from the robot to the moving obstacle.

$$\text{angle}_{\text{direction}} = \text{a}tan2((Y_{\text{obs}} - Y_{\text{robot}}), (X_{\text{obs}} - X_{\text{robot}}))$$

(*Extended rapidly exploring random tree–based dynamic path planning and replanning for mobile robots* paper)

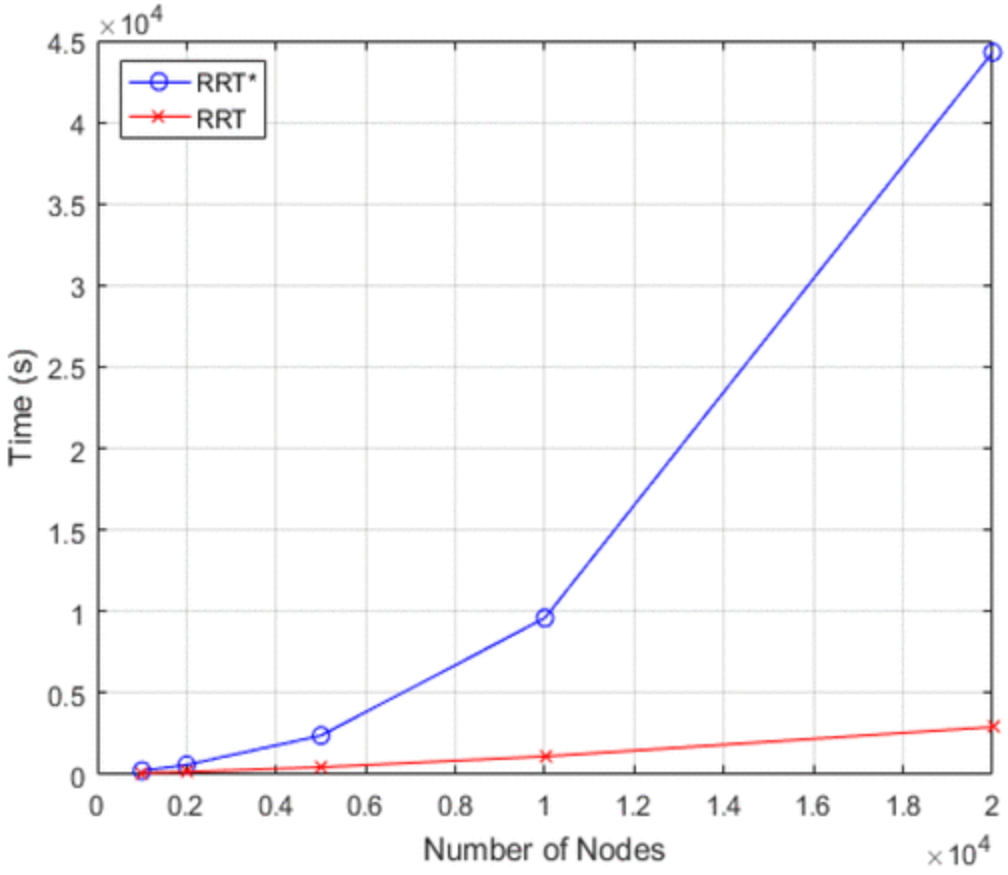
The environment will be initiated and the robots started by the central docker simulation. Each robot has its own container with individual ros2 environments.

```
T ←InitializeTree()
T ←InsertNode(∅, q_init, T)
for k ← 1 to N do
    for l ← 1 to M do
        q_rand ←RandomSample(k)
        q_nearest ←NearestNeighbor(q_rand, Q_near, T)
        q_min ←ChooseParent(q_rand, Q_near, q_nearest,
        Δq)
        T ←InsertNode(q_min, q_rand, T)
        T ←Rewire(T, Q_near, q_min, q_rand)
    end
    ShareNodes(T, k, M)
    AddSharedNodes(T, Q_shared)
    k ← M + ||Q_shared||
end
```

```
for q_node ∈ Q_shared do
    q_nearest ←NearestNeighbor(q_node, Q_near, T)
    q_min ←ChooseParent(q_node, Q_near, q_nearest, Δq)
    T ←Rewire(T, Q_near, q_min, q_node)
end
```

Although the cost for the RRT* algorithm is more consistent than RRT, the increase in the execution time with the number of nodes is very high , which makes this approach inefficient for a multi-robot problem with a huge space and high number of random obstacles.

The graph below shows the difference between the increase in execution times for RRT and RRT*, RRT is almost linearly increasing whereas RRT* is increasing exponentially



Team Members' Contributions

Talk about what each of us did… including debugging we had to spend time on

Ishwari Nalgirkar: Researched the implementations of the multi-robot problem involving different sizes of environments with different amounts of moving obstacles. RRT, RRT extended and online trajectory generator with Distributed model predictive control(paper), ROS2 implementation of the samer and Plansys and Navg2 roles in the problem. Ran the environment and the template to understand the software.

Reuben Juster: wrote up the environmental obstacles as well as helped Robert with the Pathing Localization Problems. He also developed the Pallet Storage Allocation and Collision Avoidance ideas.

Robert Ronan: Ran numerous simulations to understand the issues with the current planning system, created diagrams, wrote the current implementation of our planning function on GitHub,

and provided the sections on Robot Pathing & Localization Issues, and on The current navigation system necessitates a single sequence of paths

Sources

The project challenge
http://jderobot.github.io/RoboticsAcademy/exercises/MobileRobots/multi_robot_amazon_warehouse/

nav2_multirobot_params_X_with_control.yaml
https://github.com/JdeRobot/CustomRobots/blob/multirobot-testing/amazon_robot/amazon_robot_bringup/bringup/params/nav2_multirobot_params_1_with_control.yaml

PuLP
https://pypi.org/project/PuLP/

Robot_controller.cpp
https://github.com/JdeRobot/CustomRobots/blob/multirobot-testing/amazon_robot/amazon_robot_controller/src/robot_controller.cpp

slam-in-ros2:
http://www.robotandchisel.com/2020/08/19/slam-in-ros2/

Extended rapidly exploring random tree–based dynamic path planning and replanning for mobile robots

Devin Connell, Hung Manh La
First Published May 22, 2018 Research Article
https://doi.org/10.1177/1729881418773874

Online Trajectory Generation With Distributed Model Predictive Control for Multi-Robot Motion Planning Carlos E. Luis , Marijan Vukosavljev, and Angela P. Schoellig

http://www.dynsyslab.org/wp-content/papercite-data/pdf/luis-ral20.pdf