CURSA Research Project Final Report
Concordia University
September 2024

Research Task: Multi Robot Collaboration Considering Indoor Construction Workspace Requirements
Research Associates: Arad Hajari
Research Supervisor: Dr. Amin Hammad

1. Objective

During this research, we will develop a method for multi-robot collaboration with a specific scenario where four small robots are used to provide a safe and flexible workspace for the main robot (e.g. drilling robot) as shown in Figure 1. The location and dimensions of the workspace are automatically adjusted based on the plan of the robot and the BIM model of the floor. This dynamic workspace allocation enhances the performance of the main robot to operate seamlessly in an area where human workers are also working to ensure safety and uninterrupted operation of the robot.

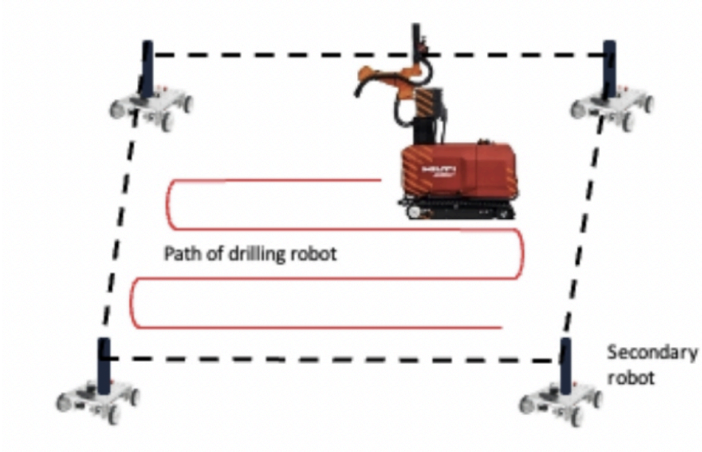

Figure 1: The robot formation we plan to achieve

The objective of this research is to explore ways of creating a multi robot system in which the robots can form formations and collaborate with each other in a meaningful way.

- The robots should be able to communicate with each other using messages
- The robots should be aware of the map of their surroundings
- The robots should be aware of the location of the other robots
- The robots should be able to navigate through the map based on the goals other robots send

- The robots should be able to form formations
- The user should be able to send goal to the master robot and slaves maintain their formation while moving to the goal
- The robots should be able to perform obstacle avoidance while maintaining the formation
- The robots should have a method for reserving the workspace to prevent moving obstacles from entering

2. Background information

ROS2 is an open-source framework designed to support the development of robotic applications. It provides the tools, libraries, and conventions to build complex and robust robot behavior across a wide variety of robotic platforms.

There are two available robots in the lab, Mecabot Pro and Mecabot, both running on ROS1. Initially the goal was to start the multi robot collaboration with these robots, but we encountered issues while trying to upgrade the robots to ROS2, and ROS1 had some issues with namespacing. Due to the explained points we have decided to apply the multi robot collaboration logic in the Gazebo Simulation before trying to do it on real robots.

Gazebo is a powerful robot simulation tool that integrates well with ROS2. It allows for the simulation of complex robotic systems in dynamic environments, providing a realistic platform for testing and development. Key features include:

- **Physics Engine**: Simulates real-world physics, including gravity, friction, and kinematics.
- **Sensor Simulation**: Provides realistic sensor feedback, such as camera images, LIDAR data, and IMU readings.
- **3D Environment**: Allows for the creation and manipulation of complex 3D environments in which robots can navigate and interact.

Gazebo can be used to simulate multiple robots simultaneously, making it an ideal tool for developing and testing multi-robot systems.

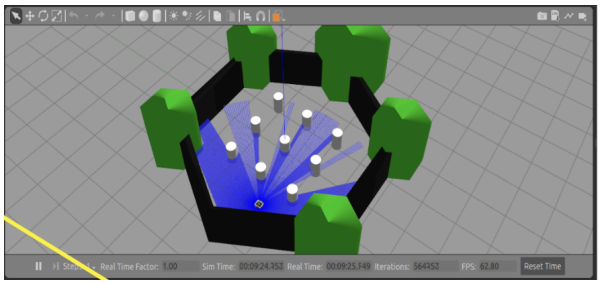

Figure 2: Example of how a robot with lidar sensor looks in Gazebo

2.1 Robot Description

In ROS2, the structure of a robot is typically defined using a combination of URDF (Unified Robot Description Format) and SDF (Simulation Description Format) files. These files describe the physical and functional aspects of the robot, including:

- **Links and Joints**: Define the robot's physical structure, specifying how different parts of the robot are connected and move relative to each other.
- **Sensors and Actuators**: Specify the types of sensors and actuators the robot has, along with their parameters and configurations.
- **Controllers**: Define the control algorithms used to drive the robot's actuators based on sensor feedback and command inputs.
- **Environment and Physics:** Define the entire simulation environment, including all objects and conditions.
- **Visual Properties:** Defines surface appearances of objects, including color, texture, and reflectivity.

By leveraging these structures, ROS2 allows for the creation of modular and reusable robot components, facilitating the development of complex multi-robot systems. In a multi-robot system, effective communication and coordination are essential for achieving collaborative tasks. ROS2 provides several tools and mechanisms to support these requirements:

- **Inter-Node Communication**: Using topics, services, and actions, robots can share information about their states, sensor data, and goals with each other.
- **TF2**: A ROS2 library that keeps track of multiple coordinate frames and allows robots to transform between them, which is crucial for maintaining awareness of each other's locations.
- **Navigation Stack**: A set of ROS2 packages that provide autonomous navigation capabilities, including path planning, obstacle avoidance, and goal setting. This stack can be extended to handle multi-robot scenarios where robots need to navigate cooperatively.

By combining these elements, a multi-robot system in ROS2 can achieve sophisticated behaviors, such as formation control, collaborative mapping, and coordinated task execution. Gazebo provides a realistic simulation environment to test and refine these capabilities before deploying them on physical robots.

2.2 ROS 2 Nodes

In ROS 2 (Robot Operating System 2), a **node** is the basic building block of the system. Nodes are independent, modular processes that perform computations. Each node is designed to perform a specific task, such as sensing, data processing, or actuation. In a multi-robot

navigation project, different nodes may handle tasks such as sensor data acquisition, localization, path planning, and motion control.

Nodes in ROS 2 are loosely coupled, meaning they can operate independently but still communicate with each other to achieve complex tasks. This modular design allows for easy scalability and flexibility, as new nodes can be added or existing nodes can be modified without disrupting the entire system.

2.3 ROS 2 Topics

Topics in ROS 2 are named buses over which nodes exchange messages. A topic is essentially a communication channel that nodes can publish to or subscribe to. For example, in a multi-robot navigation scenario:

- A sensor node might publish laser scan data on a topic named `/scan`.
- A localization node might subscribe to the `/scan` topic to receive sensor data for processing.

The communication over topics is asynchronous, meaning that nodes do not have to wait for a response from other nodes; they can continue executing their tasks. This makes the system more efficient and allows for real-time communication between nodes.

In ROS 2, topics are strongly typed, meaning that each topic has a specific message type. For example, the `/scan` topic might use the `sensor_msgs/LaserScan` message type to represent the laser scan data.

2.4 ROS 2 Actions

Actions in ROS 2 provide a mechanism for asynchronous communication that involves long-running tasks. Actions are similar to topics, but they include more structure, making them suitable for tasks that might take a significant amount of time to complete, such as moving a robot to a specific location or performing a complex manipulation.

An action in ROS 2 consists of three parts:

- **Goal:** The desired outcome, such as a target location for the robot.
- **Feedback:** Periodic updates on the progress of the action, which can be used to monitor the task without blocking other operations.
- **Result:** The final outcome once the action is completed.

Actions allow for canceling and preempting tasks, which is particularly useful in multi-robot systems where priorities may change dynamically.

2.5 Nav 2 Stack

The Nav2 (Navigation 2) Action in ROS 2 is a key component used for robot navigation. It allows a robot to autonomously navigate through an environment to a specified goal location. Nav2 builds on the concepts of ROS 2 actions and provides a framework for setting navigation goals and receiving feedback on the robot's progress.

2.6 Path Planning in Nav2

Path planning is a critical component of autonomous navigation, allowing a robot to determine the best path from its current position to a specified goal location while avoiding obstacles. In ROS 2's Navigation 2 (Nav2) stack, path planning is typically handled by the **global planner**, which calculates the optimal path based on the robot's map and the environment.

2.6.1 A* vs. Dijkstra for Path Planning

Dijkstra's Algorithm and **A*** (A-Star) are two common algorithms used for path planning. Both algorithms are graph search methods that find the shortest path from a start node to a goal node, but they do so in different ways:

2.6.1.1 Dijkstra's Algorithm

- **Approach:** Dijkstra’s algorithm is a uniform-cost search algorithm. It explores all possible paths from the start node to the goal node, ensuring that the shortest path is found by expanding nodes in all directions uniformly based on the lowest cumulative cost from the start.
- **Performance:** Dijkstra’s algorithm guarantees the shortest path but can be computationally expensive, especially in large or complex environments, because it aims to explore all nodes equally without considering the direction towards the goal.

2.6.1.2 A* Algorithm

- **Approach:** A* is an informed search algorithm that improves on Dijkstra’s by introducing a heuristic function. This function estimates the cost from the current node to the goal node, guiding the search process towards the goal more directly.
- **Heuristic function:** The most common heuristic function used in A* is the Euclidean distance or Manhattan distance, depending on the grid or space. This function helps the algorithm prioritize nodes that are closer to the goal, reducing the number of nodes it needs to explore.
- **Performance:** A* is generally faster than Dijkstra’s because it uses the heuristic function to focus the search, making it more efficient in finding the shortest path. It strikes a balance between exploring the least costly paths (like Dijkstra’s) and heading towards the goal, often leading to quicker results.

2.6.2 Using A* for Faster Path Planning in Nav2

In Nav2, the choice of path planning algorithm can significantly impact the performance of your robot, especially in dynamic or large environments. By default, Nav2 may use Dijkstra's algorithm for global path planning, but it can be configured to use A* instead.

Why Use A Instead of Dijkstra:*

- **Efficiency:** A* reduces the search space by focusing on paths that are more likely to lead to the goal, resulting in faster path planning, especially in large environments.
- **Scalability:** In complex or crowded environments, A* can provide faster responses, which is crucial for real-time navigation and dynamic obstacle avoidance.

3. Work Description

3.1 Multi-Robot Navigation in ROS 2: Understanding Namespaces and Transform Trees

In a multi-robot system using ROS 2, it's essential to organize the robots and their communication channels effectively to ensure smooth operation. Each robot needs to navigate independently, and to do so, specific guidelines must be followed:

3.1.1. Unique Namespaces for Each Robot

- **Purpose:** In a multi-robot environment, each robot operates its own set of nodes, topics, and actions and every component uses the same names for these nodes and topics. In order to prevent any conflicts between these entities, each robot is assigned a unique namespace.
- **Namespace Structure:** A namespace acts as a prefix to all the topics, nodes, and services associated with a robot. For instance:
    - **Robot 1:** `/robot_1/`
        - AMCL Node: `/robot_1/amcl`
        - Odom Topic: `/robot_1/odom`
        - Base Link: `/robot_1/base_link`
    - **Robot 2:** `/robot_2/`
        - AMCL Node: `/robot_2/amcl`
        - Odom Topic: `/robot_2/odom`
        - Base Link: `/robot_2/base_link`
- **Benefit:** This structure ensures that when Robot 1 publishes data to its `/robot_1/odom` topic, it does not interfere with Robot 2's `/robot_2/odom` topic.

Hence, they enable the reuse of ROS nodes and packages in different contexts without any name clashes. This separation provided by namespaces allows each robot to operate independently without cross-communication issues.

After applying the explained namespacing changes and spawning the multiple robots this is how a simplified version of the node trees would look like for 3 robots.

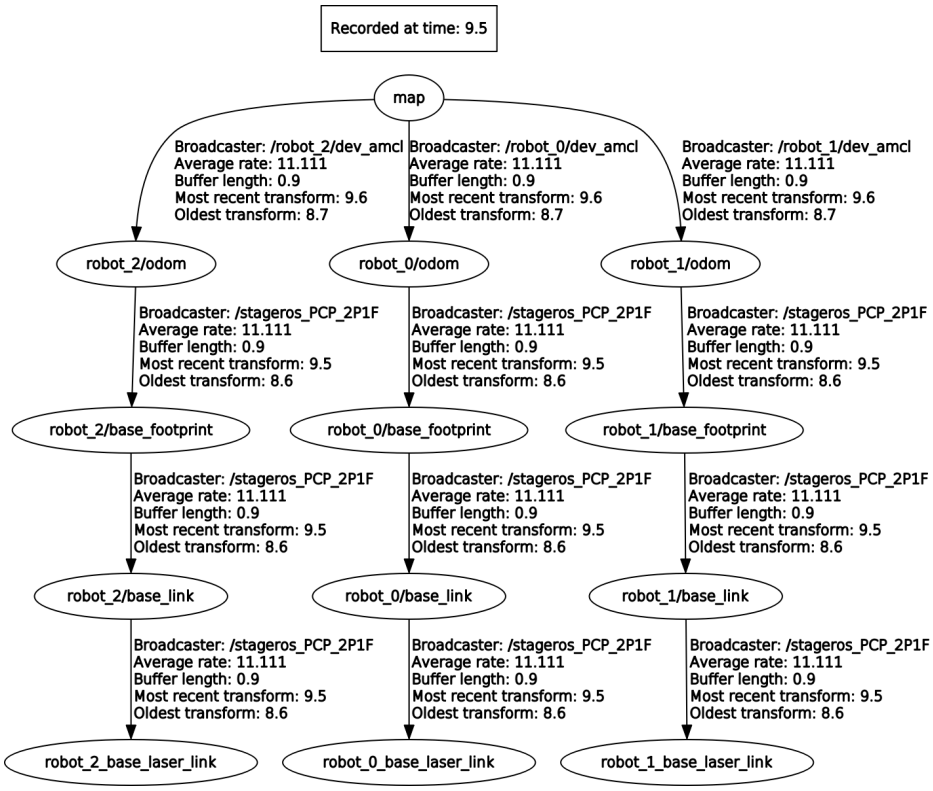

Figure 3. Node tree of a 3 robot navigation system after adding the namespaces (1)

Now the robots can coexist in the same environment with each other, the nodes for formation and collaboration should be created.

3.1.2. Output:

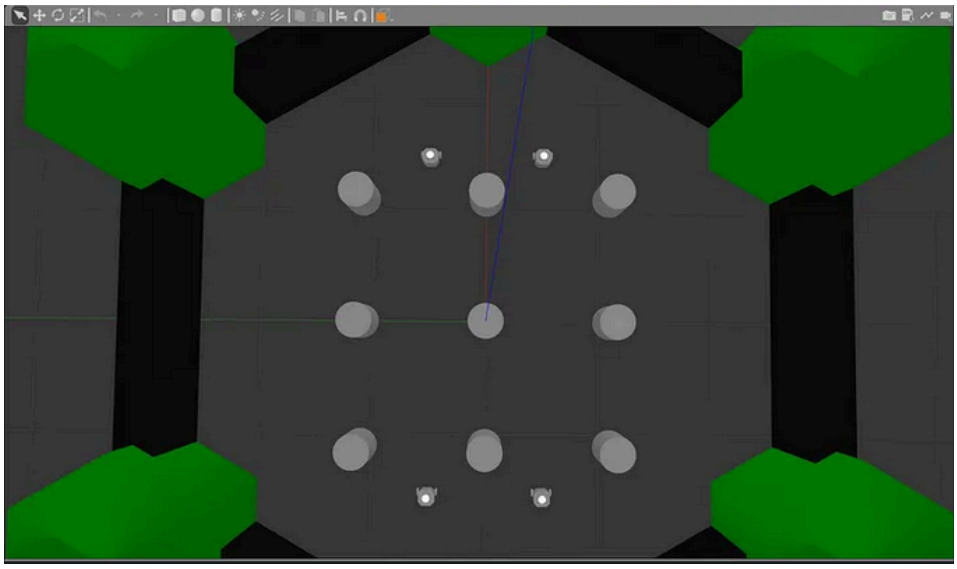

Figure 4. Four robots coexisting in an environment (2)

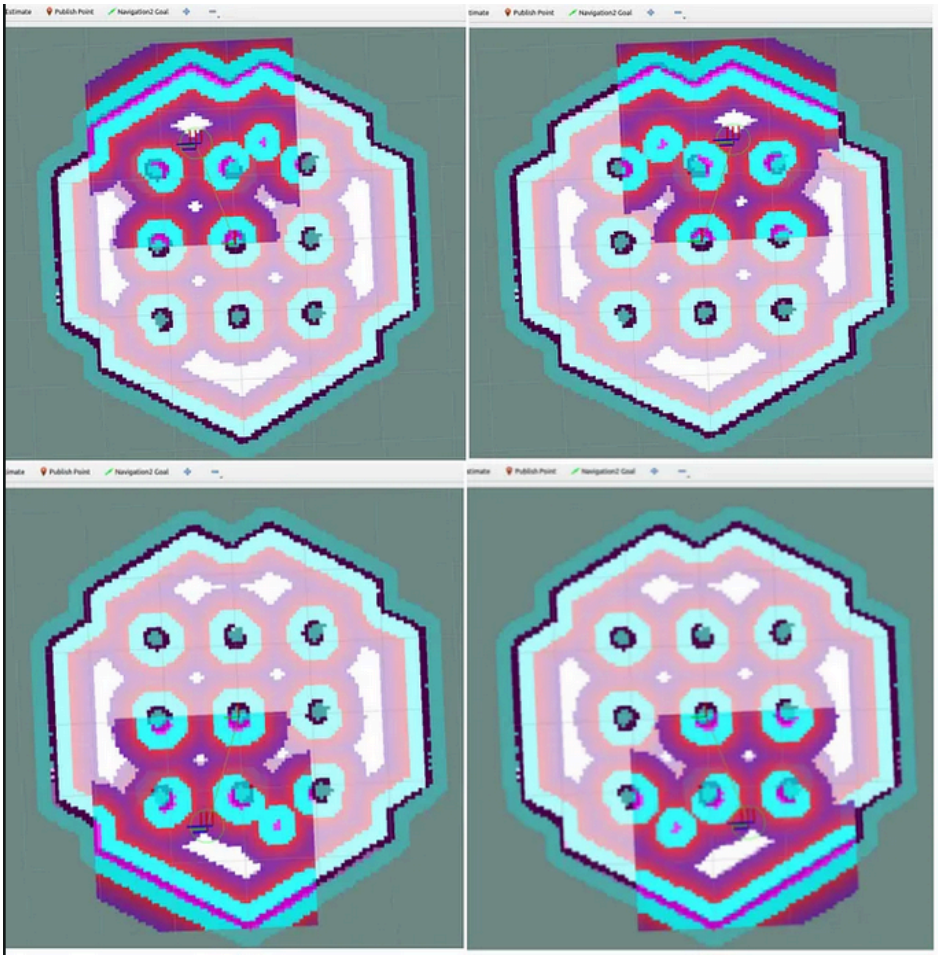

Figure 5. Lidar output and cost map of each robot shown in Rviz (2)

Global Costmap: The global costmap represents the entire known environment of the robot. It's a static map that doesn't change frequently and is used for overall path planning. In this image, the entire hexagonal shape represents the global costmap, which provides a broad view of the navigable space and obstacles.

Local Costmap: The local costmap is a smaller, more dynamic map centered around the robot's current position. It updates more frequently to account for immediate obstacles and changes in the environment. In the above image, you can see how each of the robots has its own costmap having the changing patterns within the hexagonal map representing the obstacles and the updates to the costmap.

The interplay between the stable outer structure (global costmap) and the dynamic inner patterns (local costmap) illustrates how a robot navigation system combines long-term environmental knowledge with real-time, local information to plan and execute paths safely and efficiently.

3.2. Communication and Coordination for Multi Robot Formation

To achieve meaningful collaboration between robots in a multi-robot system, it's essential to set up communication channels that allow robots to share critical information, such as their positions, status, or sensor data. This can be accomplished by creating nodes that specifically listen to topics published by other robots and perform actions based on the received data.

To achieve a simple multi-robot formation, you can implement a system where the "slave" robots continuously track the "master" robot by listening to its position and applying a formation offset. This approach ensures that the slave robots always maintain a specific relative position to the master robot, forming a predefined shape or pattern as they move.

3.2.1. Master Robot Position Broadcast

The master robot, let's call it tb1, is responsible for navigating through the environment. Its position is continuously published on the /tb1/amcl_pose topic.

This topic provides the current pose of tb1, which includes its position (x, y coordinates) and orientation (yaw angle).

3.2.2. Listening Node on Slave Robots

The slave robots (tb2, tb3, etc.) need to follow the master robot while maintaining a fixed formation. To achieve this, a node on each slave robot listens to the /tb1/amcl_pose topic.

This node continuously receives the position of the master robot (tb1) and calculates the required position for the slave robot based on a predefined formation offset.

3.2.3. Applying the Formation Offset

The formation offset defines the relative position each slave robot should maintain with respect to the master robot. For instance, if you want tb2 to follow tb1 1 meter to the right and 2 meters behind, the offset would be (x_offset, y_offset) = (1, -2).

The node on each slave robot computes the target pose by adding the offset to the pose received from /tb1/amcl_pose:

The offset is applied in the local frame of the master robot, ensuring that the formation remains consistent even when the master robot changes direction.

3.2.4. Publishing the Commanded Velocities

Once the target pose for the slave robot is calculated, it needs to be translated into velocity commands that will drive the robot towards the desired position.

The node publishes the appropriate velocity commands (cmd_vel) to the slave robot's control topic, typically /tb2/cmd_vel for tb2, ensuring that the robot moves in such a way as to maintain the formation:

These commands adjust the robot’s linear and angular velocities to align with the target pose that was calculated using the offset.

3.2.5. Maintaining Formation

As the master robot (tb1) moves, the slave robots continuously update their target positions based on the updated amcl_pose of the master.

By repeatedly applying the formation offset and publishing the resulting velocity commands, the slave robots maintain the desired formation relative to the master robot.

Using this approach the robots will form a static formation and keep it under any circumstances. This is good but there is a big problem, what if there is an obstacle in front of one of the robots while moving, a collision will happen!

3.3. Multi Robot Formation with Obstacle Avoidance

In the previous example, the method of building a simple robot formation was explained, now how to add obstacle avoidance to this formation?

Incorporating obstacle avoidance into a multi-robot formation requires integrating the robots' movement with the ROS 2 Navigation Stack, which handles path planning and obstacle avoidance. By leveraging the navigation stack, each robot can independently avoid obstacles while maintaining its position within the formation.

3.3.1. Overview

To achieve a multi-robot formation with obstacle avoidance:

1. **Master Robot (`tb1`) Receives Goal:** The master robot receives a goal position through the `/tb1/goal_pose` topic.
2. **Formation Offset Calculation:** A dedicated node listens to the goal sent to the master robot, calculates the formation offsets for the slave robots, and publishes new goal positions to each slave robot's respective `/tbX/goal_pose` topic.
3. **Navigation Stack Execution:** Each robot uses the ROS 2 Navigation Stack to navigate to its assigned goal while independently avoiding obstacles based on the current map, global costmap, and local costmap.

To achieve a multi-robot formation with obstacle avoidance, a dedicated node should be created that constantly listens for goal positions sent to the master robot (`tb1`). When a goal is sent to `/tb1/goal_pose`, this node listens, applies the necessary offsets for the slave robots, and then publishes the adjusted goal positions to each corresponding slave robot. For example, if a goal position is sent to `/tb1/goal_pose`, the node will listen to this, calculate the offset positions, and then publish the new goals to `/tb2/goal_pose`, `/tb3/goal_pose`, `/tb4/goal_pose`, and `/tb5/goal_pose`. This ensures that all five robots will move towards their respective goals with obstacle avoidance, maintaining the desired formation as they reach their destination.

This approach allows multiple robots to navigate as a cohesive unit, maintaining a specific formation while independently avoiding obstacles. The master robot’s goal is used as the reference point, with the formation offset node ensuring that each slave robot follows suit by receiving its adjusted goal position. The ROS 2 Navigation Stack, with its path planning and obstacle avoidance capabilities, ensures that each robot reaches its destination safely while preserving the formation structure. This system is particularly useful in complex environments where both coordination and autonomy are crucial for successful multi-robot operations.

3.4. Reserving the Workspace once the formation reaches the goal position

To achieve the goal of reserving the workspace around the slave robots and preventing any moving obstacles from entering this area, a specialized node can be developed. This node will listen to each robot's `/goal_pose` (the target position) and `/amcl_pose` (the current position). The node monitors these positions and, once all five robots have reached their respective goals, it can trigger preventative actions to secure the workspace.

3.4.1. Implementation Details

3.4.1.1. Listening to Goal and Current Positions

- The node will subscribe to the `/goal_pose` and `/amcl_pose` topics for each robot (e.g., `/tb1/goal_pose`, `/tb1/amcl_pose`, `/tb2/goal_pose`, `/tb2/amcl_pose`, etc.).
- By continuously monitoring these topics, the node can track each robot's progress towards its goal.

3.4.1.2. Detection of Goal Achievement:

- The node will determine when all robots have reached their goal positions by comparing the `/amcl_pose` (current position) with the `/goal_pose` (target position) for each robot.
- When all robots are within a predefined tolerance of their goals, the node will recognize that the formation has been achieved.

3.4.1.3. Preventative Actions to Secure Workspace:

- Once all robots have reached their goals, the node can implement preventative measures to protect the reserved workspace around the robots.
- Example Preventative Measure: A virtual box or barrier can be spawned around the robots to prevent any outside objects or obstacles from penetrating the workspace. This box can be implemented using ROS 2's simulation tools or through control commands that modify the environment.
- The spawning of the boundary walls is triggered when the system detects that all robots have reached their goal. This detection is typically implemented through a state-checking mechanism, where each robot reports its status back to a central node. When the trigger condition is met, the central node sends a command to spawn the boundary walls. This is done using ROS 2’s Gazebo simulation

environment, where a service call is made to instantiate the URDF-defined walls at specific coordinates around the robots. The walls' positions are dynamically calculated based on the robots' current positions and the desired formation. This ensures the walls are appropriately sized and placed to form a complete enclosure.

- **Alternative Actions:** Other preventative actions could include triggering alarms, altering the robots' positions to block entry points, or dynamically adjusting the robots' formation to cover more ground.
- **Deletion of the boundary walls on receiving a new goal:** The deletion of the walls occurs when a new goal is assigned to the master robot. Before the robots start moving towards the new goal, the walls must be removed to allow free movement. The central node issues a command to delete the existing walls. In ROS 2, this is typically done by calling a service to remove the spawned objects from the simulation environment. The process ensures that the workspace is cleared before the robots start their next task. The transition between wall spawning and deletion is handled carefully to avoid any delays or interruptions in the robots' tasks. The system checks that the walls are fully deleted before issuing movement commands to the robots.

By creating this node, the system not only ensures that all robots maintain their formation but also protects the reserved workspace from external interference. The ability to spawn a virtual box or implement other preventative measures provides flexibility in how the system secures the area, making it adaptable to different environments and challenges. This approach enhances the safety and reliability of multi-robot operations, especially in dynamic and unpredictable environments.

The final version of the project and the videos of the most recent tests can be found in this github repository. (3)

4. Plans for Future

In the future, several enhancements can be made to achieve the goal of having four robots form a workspace while a fifth robot performs drilling inside the reserved area:

1. **Infrared Laser Barriers:** Instead of creating a physical or virtual box to secure the area, infrared lasers could be installed between each pair of robots. If the laser is interrupted, a visual or sound alert can be triggered to prevent intruders from entering the workspace.
2. **Queue Formation Movement:** Currently, the robots navigate separately towards their formation goals. An improvement would be to implement a queue formation, where the robots move behind each other in a line. This would make it easier for humans to avoid robots in crowded environments.

To implement this method a node should be created which sends the current location of the robot1 in the queue as a goal to robot2, sends the location of robot2 as a goal to robot3, … this process should be repeated every 2 seconds.

3. **Multiple Control Levels:** The system could be enhanced by implementing multiple control levels—one dedicated to formation and workspace reservation, another for assigning the drilling points, and a third for executing the drilling task. This would provide a more structured and efficient approach to managing the different aspects of the operation.

(1) https://answers.ros.org/upfiles/14918813927828759.png
(2) https://medium.com/@arshad.mehmood/a-guide-to-multi-robot-navigation-utilizing-turtlebot3-and-nav2-cd24f96d19c6
(3) https://github.com/AradHajari/Concordia-Research-Multibot