EECE 5554 Robotic Sensing & Navigation

Project Report

SYNCHRONIZATION AND COORDINATION OF TWO MOBILE ROBOTS

Group 6

Presented by: Siyu Liu, Jorge Ortega, Ahilesh Vadivel and Sachidanand Halhalli

1. Introduction

The rapid advancement of robotics has led to the deployment of mobile robots across various sectors, including logistics, search and rescue, and environmental monitoring. A critical capability for these robots is Simultaneous Localization and Mapping (SLAM), which enables them to navigate and map unknown environments autonomously. While single-robot SLAM systems have achieved significant success, they often face limitations in efficiency and scalability when addressing large or complex areas. To overcome these challenges, multi-robot SLAM systems have been developed, allowing multiple robots to collaboratively explore and map environments, thereby enhancing efficiency and robustness.

This project investigates the synchronization and coordination of two mobile robots tasked with independently executing SLAM, exploration, and navigation within a shared environment, followed by the merging of their respective maps. This collaborative approach not only accelerates the mapping process but also improves the accuracy and resilience of the generated maps. The implementation utilizes the Robot Operating System (ROS) for integration and communication, Gazebo for simulation, SLAM algorithms for localization and mapping, and RViz for visualization. By leveraging these tools, the project aims to develop a robust framework for multi-robot collaboration in mapping tasks

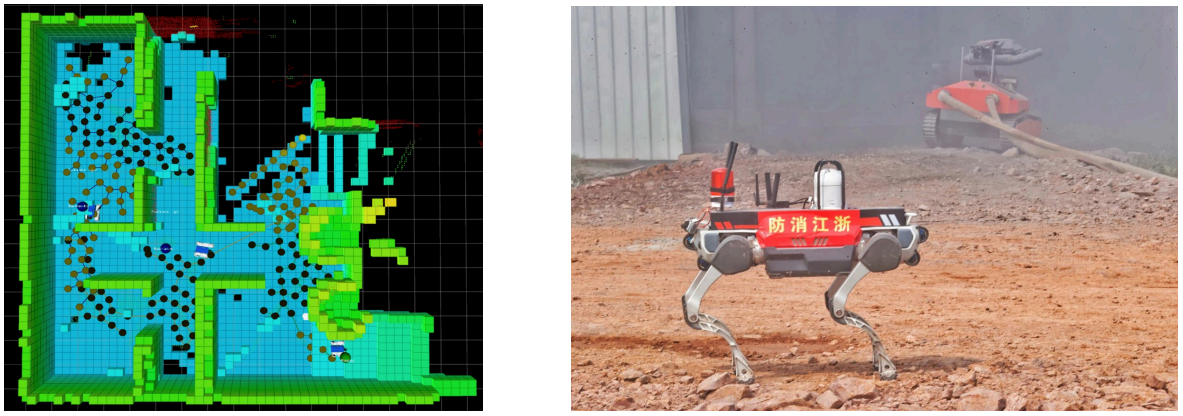


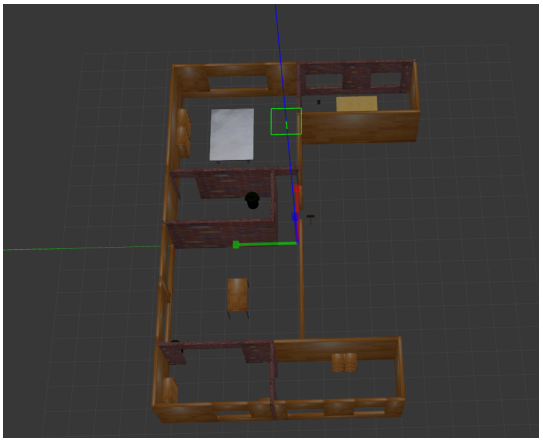
Figure (1): [Left] Simulation application of multi-robot coordination [Right] Robot dog detecting explosive hazards

2. Work method

As mentioned before this project used Gazebo as the primary simulation environment. This open-source simulator is a high-fidelity simulation environment with realistic physics, sensor feedback, and a wide range of supported robot models. Additionally, the original Gazebo ROS packages include a large variety of pre-built works. These greatly facilitated initial testing and experimentation for future setup and modifications to adapt the world to project's specific goals. Apart from the Gazebo ROS package, the *TurtleBot3* ROS package resulted very useful for the development of the project. This package provided us with multiple predefined TurtleBot3 robots and various functions like SLAM, Monte Carlo localization and teleoperation of the *TurtleBot3* robots.

To streamline all the project files and manage the dependencies, a custom ROS workspace was created and shared between the group members. Within this workspace, our launch and configuration files were developed based on the thoroughly studied *TurtleBot3* files, adapting them to accommodate the mapping and navigation functions to the 2 robots defined in our personalized Gazebo world.

Figures (2) and (3) represent the Gazebo world and the TurtleBot3 robots used in the simulations.



Figure(2): Gazebo world used for the project.

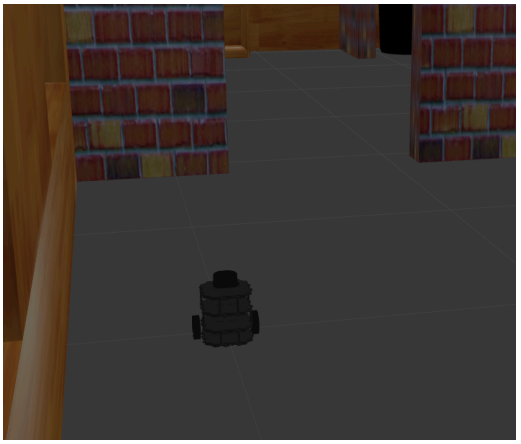


Figure (3): TurtleBot3 model used.

To avoid conflicts in a multi-component setup, the launch files in the current project were coded using namespaces. This tool is used in ROS to ensure that there are no conflicts between the nodes and topics that run simultaneously. The following figure shows how namespaces are included in the file that launches both Turtle robots in the Gazebo world.

```
<group ns = "$(arg first_tb3)">
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
  <param name="tf_prefix" value="$(arg first_tb3)" />
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher" output="screen">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" value="$(arg first_tb3)" />
  </node>

  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -model $(arg first_tb3) -x $(arg first_tb3_x_pos) -y $(arg first_tb3_y_pos) -z
</group>

<group ns = "$(arg second_tb3)">
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
  <param name="tf_prefix" value="$(arg second_tb3)" />
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher" output="screen">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" value="$(arg second_tb3)" />
  </node>

  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -model $(arg second_tb3) -x $(arg second_tb3_x_pos) -y $(arg second_tb3_y_pos)
</group>
```

Figure (4): Use of namespaces to spawn turtlebots.

Once both robots were correctly spawned in different locations of the simulated environment, the SLAM node was initialized for each robot. Gmapping was the ROS package set as the SLAM method to provide a 2-Dimensional map in real time of the data scanned by each robot. Afterwards, a full map of the Gazebo world was successfully obtained by tele operating both robots at the same time. This process is visible in figures (5) and (6).

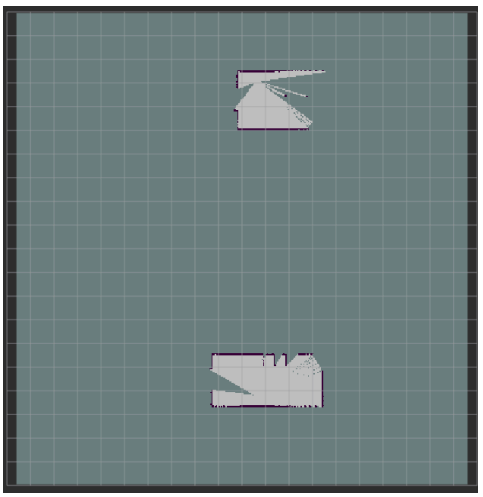


Figure (5): Representation of individual maps

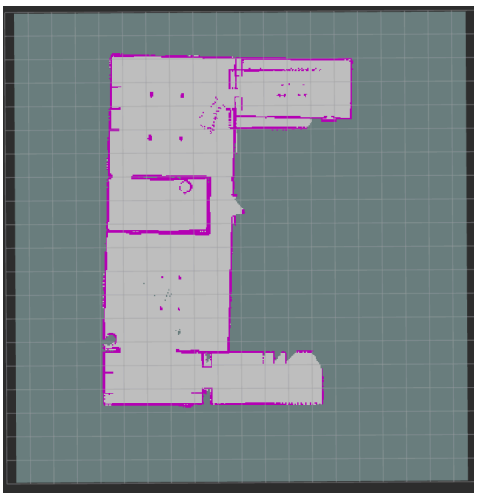
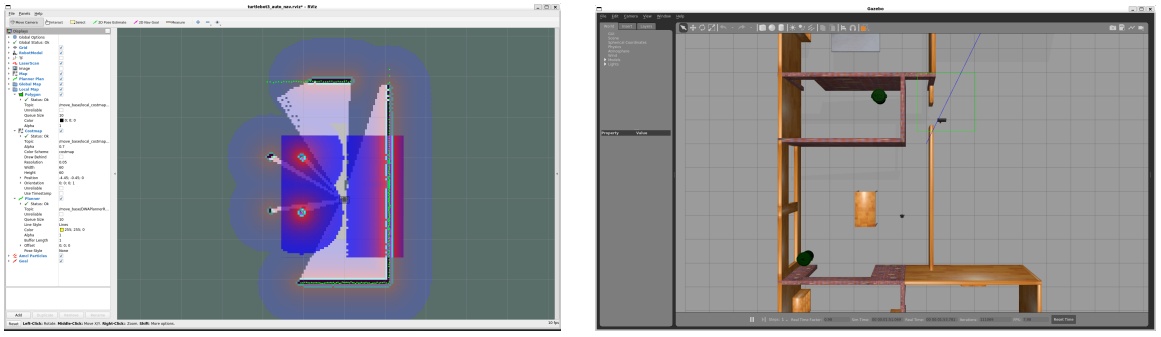


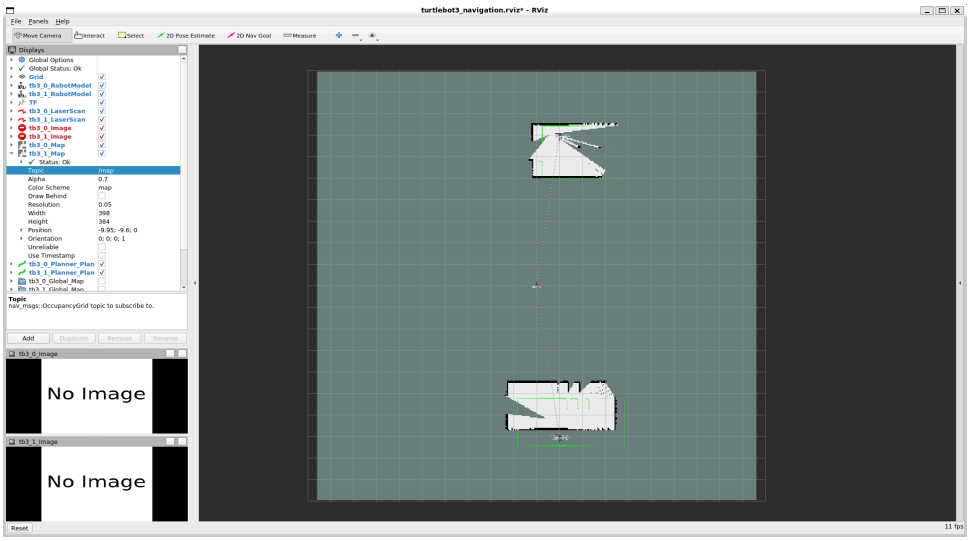
Figure (6): Full map of the Gazebo world

To implement autonomous navigation for the turtle robots, the *amcl_package*, *move_base* package, and *slam_package* were utilized together to enable navigation based on dynamically received SLAM data. The initial step involved testing the autonomous navigation of a single robot, where the SLAM data provided a map of the robot's environment. This data was then fed into AMCL (Adaptive Monte Carlo Localization), which generated localization data along with laser data. The combined information was subsequently passed to the *move_base* package, which created a path plan to reach a specific goal position or destination. The figures below depict how the system views the unknown environment simulated in the Gazebo world.



Figure(7): [Left]Visualisation of SLAM and Laser sensor data along with local map, costmap and global map data. [Right] Gazebo simulation environment with a single turtle robot.

To visualize the data acquired from the robot's sensors, RViz (ROS Visualization) was employed as a graphical interface. This tool allowed for the visualization of the local map, global map, and costmap derived from SLAM and laser data, facilitating a central map that merged information from two different turtle robots. The following figure shows an example of the RViz tool used to visualize the scan and laser data obtained by the two turtle robots used in the current project.

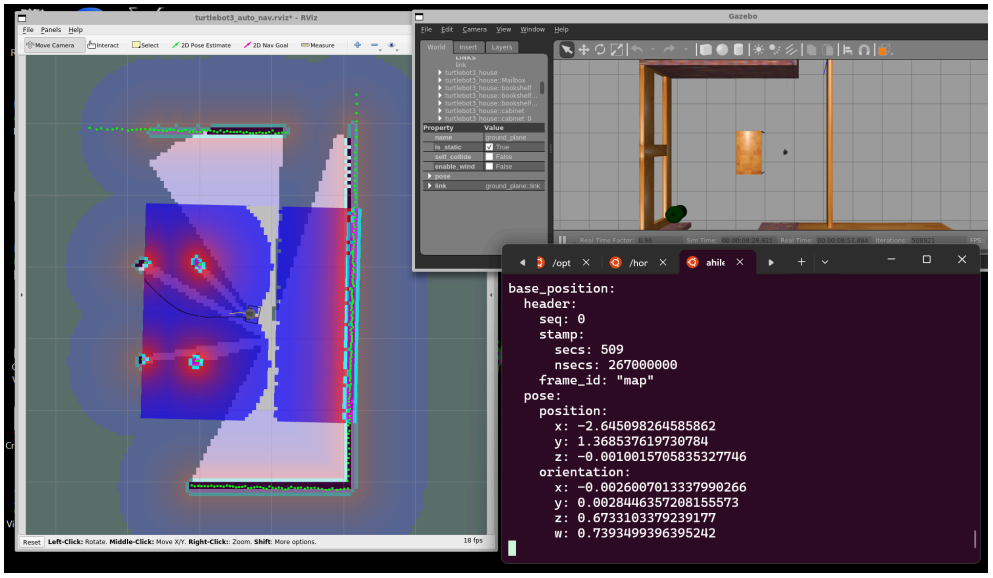


Figure(8): Visualization of 2 turtle robots using SLAM and Laser data to map the environment.

The next step was to implement autonomous navigation for two turtle robots, which required separate name spaces for each robot. This necessitated individual instances of the *amcl*, *move_base*, and *slam* packages for both robots, as well as a new launch file to accommodate these modifications. Additionally, the *file.rviz* required updates concerning sensor parameters, map data parameters, TF tree parameters, and other relevant settings to ensure proper functionality and visualization of the multi-robot system. The above figure depicts the laser data and SLAM data collected by 2 turtle robots and merging into a central map.

3. Results

The autonomous navigation using SLAM and laser data for a single turtle robot was successfully implemented, enabling comprehensive environment data recording. Through AMCL (Adaptive Monte Carlo Localization), the boundaries of the environment were precisely mapped, and detailed information about the robot's distances from obstacles was generated. The *move_base* package complemented this process by providing advanced map planning capabilities, which effectively facilitated the robot's navigation and movement towards its designated goal position.



Figure(9): Movement of a single turtle robot using the path generated from move_base package using the data from AMCL and SLAM.

In the attempt of launching two robots in the same simulated world, multiple significant challenges were faced in correlation to TF frames. Both robots were publishing transforms with identical names, such as *base_footprint* and *odom*, which led to a constant *TF_REPEATED_DATA* error. This error caused the TF tree to break down, as the system was unable to distinguish between the two robots. As a result, the transforms from one robot were not being updated correctly, which negatively impacted synchronization and overall system performance.

To resolve this, we grouped each robot in its own namespace as it was described earlier. This approach isolated the robots, allowing them to be treated as separate entities within the same simulation environment. By doing so, we successfully enabled both robots to coexist and operate independently in the same world.

Moving on, we applied the mapping algorithm to both robots, aiming to generate individual maps and then merge them into a complete map. This was done through a *map_merge* file that read from the scan topics from both robots and published the merged map through a new general topic called */map*.

A fraction of this file is represented in figure (11).

```
  <node pkg="multirobot_map_merge" type="map_merge" respawn="false" name="map_merge" output="screen">
    <param name="robot_map_topic" value="map"/>
    <param name="robot_namespace" value="tb3"/>
    <param name="merged_map_topic" value="map"/>
    <param name="world_frame" value="map"/>
    <param name="known_init_poses" value="true"/>
    <param name="merging_rate" value="0.5"/>
    <param name="discovery_rate" value="0.05"/>
    <param name="estimation_rate" value="0.1"/>
    <param name="estimation_confidence" value="1.0"/>
  </node>

  <node pkg="tf" type="static_transform_publisher" name="world_to_$(arg first_tb3)_tf_broadcaster" args="0 0 0 0 0 0 /map /$(arg first_tb3)/map 100"/>
  <node pkg="tf" type="static_transform_publisher" name="world_to_$(arg second_tb3)_tf_broadcaster" args="0 0 0 0 0 0 /map /$(arg second_tb3)/map 100"/>
</launch>
```

Figure (10): setup of themap_merge file.

Regardless of these adjustments, we continued encountering miscommunication issues between *slam_gmapping* and *explore_lite*. The TF frames published by *slam_gmapping* (such as the map → *base_footprint* transform) were not being properly related to *explore_lite*, causing the mapping frames to be missed by the exploration node. This communication failure hindered the autonomous exploration process.

Despite our efforts to identify the node responsible for this miscommunication, we were unable to pinpoint the source of the issue. Each node was meticulously reviewed and a deep research was carried out in order to solve the error. Consequently the goal defined in the project proposal regarding the communication between both robots was not able to be reached.

4. Conclusion and future work

Through this project, several critical lessons about managing multiple robots in a simulation environment were acquired. Specifically, a deeper understanding of the importance of namespace management in ROS to prevent TF conflicts was obtained. The use of distinct namespaces allowed us to avoid issues where the system couldn't differentiate between the two robots' TF frames.

Moreover, we learned how communication between different nodes, such as *slam_gmapping* and *explore_lite*, is crucial for successful autonomous exploration. Misconfiguring the topics or publishing incorrect transforms can prevent nodes from receiving the necessary data, ultimately leading to failures in tasks like map generation and exploration.

Despite the challenges faced, the team was able to generate individual maps for each robot and conduct exploration tasks. However, the system's performance could be improved by addressing the communication issues and enhancing the synchronization between the robots and exploration nodes.

In the future, improvements could include refining the TF management system and ensuring better synchronization between nodes. Additionally, expanding the system to handle more robots would require careful planning of topic remapping and frame publishing to ensure smooth collaboration among the robots.

5. References

1. Dudek, G., Jenkin, M., Milios, E., & Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots, 3*(4), 375-397. https://doi.org/10.1007/BF00140817
2. Quigley, M., Gerkey, B., & Smart, W. D. (2009). *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media.
3. “TurtleBot3 Simulation.” *TurtleBot3*, https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/. Accessed 11 December 2024.