Multi-Robot Navigation System

Author:	Caesar (Zhiye) Zhao
Institution:	University of Technology Sydney (UTS)
Supervisor:	Dr. Felix Kong
Project Type:	Team Research & System Implementation
Date:	March 2025

Abstract:

This report presents the design, implementation, and evaluation of a multi-robot navigation system. The system integrates collaborative path planning, dynamic obstacle avoidance, and decentralized control. It utilizes Simultaneous Localization and Mapping (SLAM), Dijkstra's algorithm for global path planning, and Model Predictive Control (MPC) for local trajectory generation. The system's effectiveness is validated through both simulation and physical robot experiments.

Table of Contents

Introduction

Project Background and Objectives

Multi-robot systems are becoming increasingly important in modern industrial and service applications, particularly in warehouse logistics, agricultural automation, disaster response, and security. Multi-robot path planning is a crucial technology to ensure these robots efficiently complete their tasks. The goal of path planning is to find an optimal path for each robot from its starting position to its destination while avoiding obstacles and other robots. Achieving this goal requires a combination of global path planning and local path planning methods to ensure efficient navigation in complex and dynamic environments (Lin et al., 2022).

This project aims to design and implement a multi-robot path planning system using the Dijkstra algorithm for global path planning and the Dynamic Window Approach (DWA) for local path planning. Additionally, the initial design includes an exploration of the A* algorithm and the Artificial Potential Field (APF) method to investigate their performance and suitability for path planning tasks (Chen et al., 2020; Zhao & Zhu, 2011; Du & Nan, 2016).

Brief introduction to the Overall System Architecture

This project aims to design and implement a multi-robot path planning system using the Dijkstra algorithm for global path planning and the Dynamic Window Approach (DWA) for local path planning. Additionally, the initial design includes an exploration of the A* algorithm and the Artificial Potential Field (APF) method to investigate their performance and suitability for path planning tasks (Madridano et al., 2021).

The entire multi-robot path planning system consists of several key modules:

1. **Task Assignment Module**: Responsible for assigning tasks to robots based on their current states and task priorities, ensuring that each robot can reasonably complete its assigned tasks.
2. **Path Planning Module**: Comprises global and local path planning submodules. Global path planning uses the Dijkstra algorithm to determine the optimal path from the start position to the goal position, while local path planning uses the DWA algorithm to handle real-time obstacle avoidance, ensuring the robot's safe movement in a dynamic environment.
3. **SLAM (Simultaneous Localization and Mapping) Module**: Utilizes Adaptive Monte Carlo Localization (AMCL) to create and update the environment map while accurately tracking each robot's position.

Through the collaborative operation of these modules, the multi-robot system can efficiently complete path planning and task execution in complex environments. This project not only implements the above functional modules but also ensures the system's stability and reliability through detailed testing and validation, laying a solid foundation for the deployment of multi-robot systems in real-world applications.

Contract and System Architecture

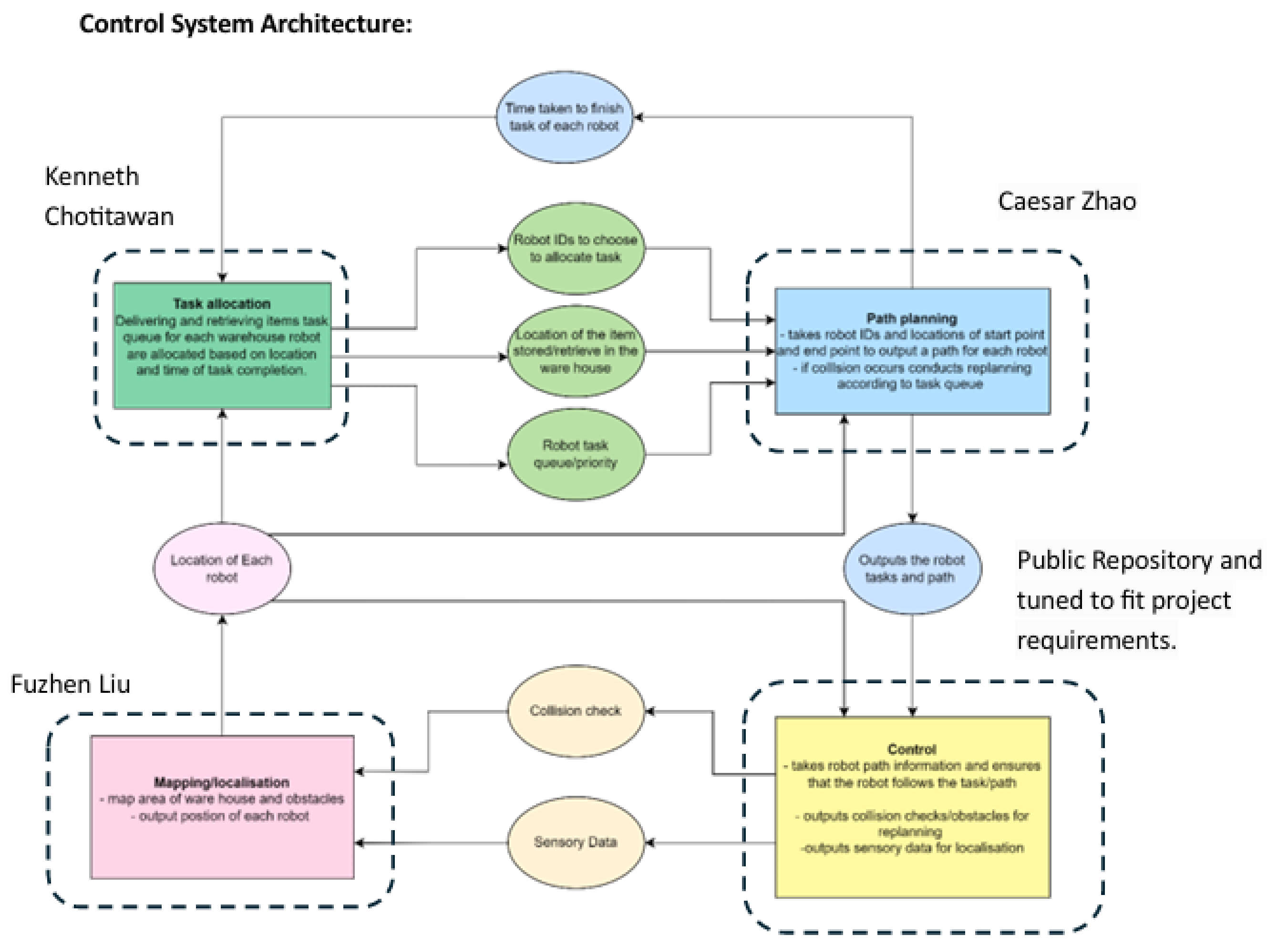
Requirements and Contract from Review 1

Trajectory planning: Implement dynamic integration of global and local path planning in a multi-robot system, aiming to explore and optimize the optimal route in real time.

- F: PRM, Dijkstra/A* with no collision checking against other robot
- P: PRM, Dijkstra/A* with post-processing collision checking against other robot
- C: Any kind of planning that collision checks with other robot during the moving stage
- D: Plus, path smoothing to make it so that robots don't have to stop and spot turn.
- HD: Integrate local path planning and global path planning to find the global optimal path and adjust the local optimal path in real time to complete complex tasks.

System Architecture Details

The system architecture consists of three main parts: task assignment, multi-robot path planning, and multi-robot SLAM. The architecture is designed to ensure efficient task allocation, path planning, and real-time environment mapping and updates. Figure 1 in the system architecture diagram illustrates the details and interactions of each module.



Control System Architecture:
Time taken to finish task of each robot
Kenneth Chotitawan
Caesar Zhao
Robot IDs to choose to allocate task
Task allocation
Delivering and retrieving items task queue for each warehouse robot are allocated based on location and time of task completion.
Location of the item stored/retrieve in the ware house
Path planning
- takes robot IDs and locations of start point and end point to output a path for each robot
- if collision occurs conducts replanning according to task queue
Robot task queue/priority
Location of Each robot
Outputs the robot tasks and path
Public Repository and tuned to fit project requirements.
Fuzhen Liu
Collision check
Mapping/localisation
- map area of ware house and obstacles
- output postion of each robot
Sensory Data
Control
- takes robot path information and ensures that the robot follows the task/path
- outputs collision checks/obstacles for replanning
-outputs sensory data for localisation

Figure 1

1. **Task Assignment Module:**

    - **Function:** Delivers and retrieves items task queue for each warehouse robot based on location and task completion time.
    - **Inputs:** Time taken to finish tasks, robot IDs, location of items in the warehouse, robot task queue/priority.
    - **Outputs:** Robot IDs for task allocation, task details including start and end points.
2. **Path Planning Module:**
    - **Function:** Takes robot IDs and locations of start and end points to output a path for each robot. Conducts replanning if a collision occurs according to the task queue.
    - **Inputs:** Robot IDs, locations of start and end points, task queue.
    - **Outputs:** Robot tasks and paths.
3. **SLAM (Simultaneous Localization and Mapping) Module:**
    - **Function:** Maps the area of the warehouse and obstacles, outputs the position of each robot.
    - **Inputs:** Collision check, sensory data.
    - **Outputs:** Location of each robot, updated maps.
4. **Control Module:**
    - **Function:** Ensures robots follow the planned paths and handles collision checks/obstacle avoidance. Provides sensory data for localization.
    - **Inputs:** Path information from the path planning module, sensory data.
    - **Outputs:** Control commands for robots, updated sensory data.

System Integration:

- The **Task Assignment Module** assigns tasks to robots and provides initial path details to the **Path Planning Module**.
- The **Path Planning Module** calculates the optimal paths using global path planning (Dijkstra/A*) and adjusts the paths in real-time using local path planning methods (DWA/APF).
- The **SLAM Module** continuously updates the environment map and robot positions, providing necessary data for path replanning.
- The **Control Module** ensures the robots follow the planned paths and adjusts for any dynamic obstacles or changes in the environment.

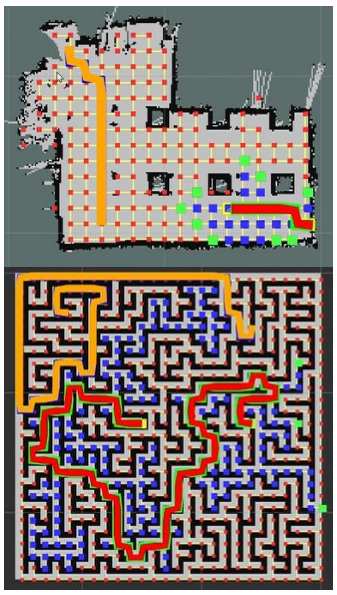
The integrated system allows for efficient multi-robot coordination and dynamic path planning, enabling robots to complete complex tasks in real-time while avoiding collisions and optimizing their routes.

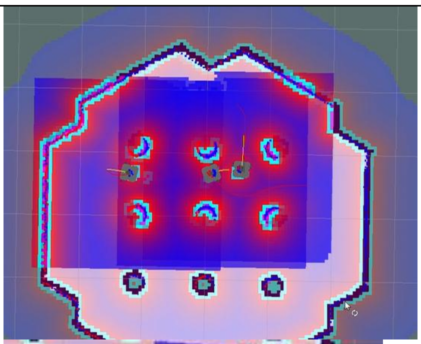
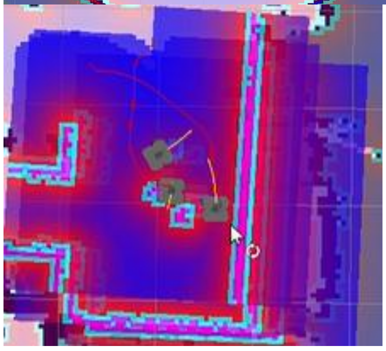
Test Plans and Evidence of Completion

In this section, we will detail the test plans and evidence of completion for each phase of the project. The test plans cover various aspects, from subsystem testing to system integration testing, ensuring the functionality, reliability, and performance of the system. The following are the specific test plans and results.

Subsystem Test Plan

First, we conducted tests on the path planning subsystem. These tests aimed to verify the effectiveness and safety of the path planning system in a multi-robot warehouse environment.

Task	State	Evidence
Test 1: Path Generation and Following • **Requirement**: Verify the system can generate optimal paths for each robot to reach designated destinations. • **Test Procedure**: In the simulation environment, assign a destination for each robot, have the system generate paths, and make the robots follow these paths. • **Evaluation Criteria**: All robots reach their destinations without collisions, following the optimal paths. • **Additional Resources**: High-precision warehouse map from SLAM. • **Input:** map, robot and target position • **Output:** path without collisions with obstacles • **Software:** Rviz in ROS or Python • **Language:** Python • **Algorithm:** A*	**Pass**	 **Photo:** These two photos show the paths generated by two robots using A* in the laboratory map and the maze map, respectively. **Video recording:** Please watch “Astra. mp4” **Explanation:** The photos show the A* algorithm's path generation process for robots in different environments. The top image is the laboratory map, and the bottom is the maze map. Both images visualize the A* algorithm's process , meeting the test criteria.

Test 2: Dynamic Obstacle Avoidance • **Requirement**: Verify the system can replan paths when encountering unexpected obstacles. • **Test Procedure**: Introduce obstacles randomly on the path of a robot during task execution and observe the system's response. • **Evaluation Criteria**: Robots can replan their paths to avoid them and continue their tasks. • **Additional Resources**: Dynamic obstacle simulator. • **Input:** map, robot and target position, dynamic model • **Output:** path without collisions with obstacles and dynamic model, simulation that the robot can go through the map has the dynamic model • **Software:** Rviz in ROS or Python • **Language:** Python • **Algorithm:** APF	**Pass**	 **Photo:** These two photos show three robots performing path planning in a laboratory map and another map, respectively. **Video recording:** Please watch “multi_robot_navi. mp4” and “multi_navigation. mp4” **Explanation:** The photos illustrate the global and local path planning processes for three robots as part of the dynamic obstacle avoidance test. The top image shows the robots navigating in a laboratory map, while the bottom image shows them navigating in another map. Both images visualize how the robots adapt their paths in response to dynamic obstacles, demonstrating the effectiveness of the algorithm used in the test.

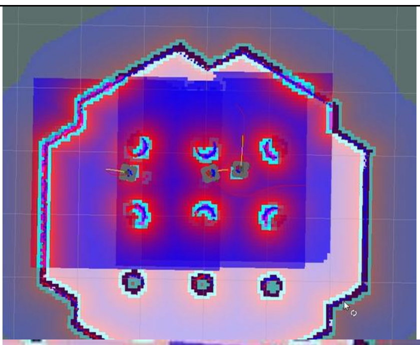
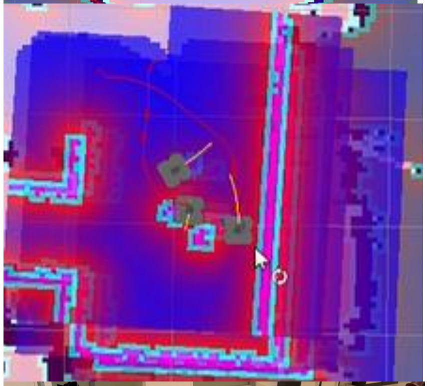
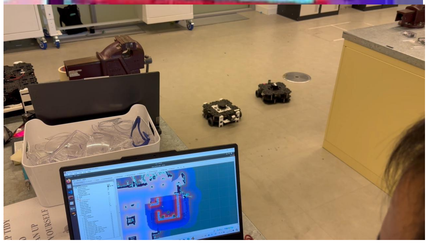
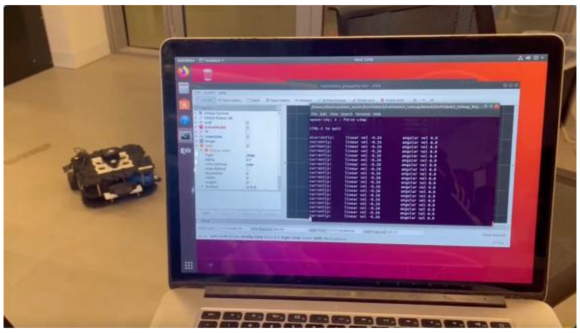
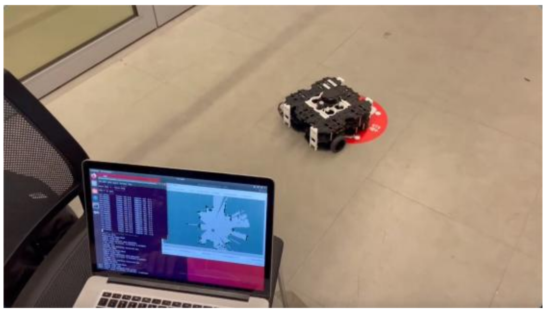
Test 3: Multi-Robot Coordination • **Requirement**: Verify the system can coordinate actions among multiple robots to avoid collisions and optimize overall path efficiency. • **Test Procedure**: Deploy multiple robots in the simulation environment to perform different tasks simultaneously, observing the coordination mechanism between robots. • **Evaluation Criteria**: Robots can coordinate with each other to avoid collisions while executing tasks and optimize overall path efficiency as much as possible. • **Additional Resources**: Command from Multi-task allocation • **Input:** map, command (robots and targets position) • **Output:** path without collisions with obstacles and each robot, a simulation that the robot can go through the map has the dynamic model and multi-robot • **Software:** Rviz in ROS or Python • **Language:** Python • **Algorithm:** APF/DWA	**Pass**	**Photo:** These two photos show three robots performing path planning in a laboratory map and another map, respectively. **Video recording:** Please watch "multi_robot_navi. mp4", "multi_navigation. mp4" and "Multi-robot_dynamic_obstacl_avoidance.MOV" **Explanation:** The photos illustrate the global and local path planning processes for three robots as part of the multi-robot coordination test. The top image shows the robots navigating in a laboratory map, while the bottom image shows them navigating in another map. Both images visualize how the robots coordinate their paths to avoid collisions and optimize overall path efficiency, demonstrating the effectiveness of the algorithm used in this test.

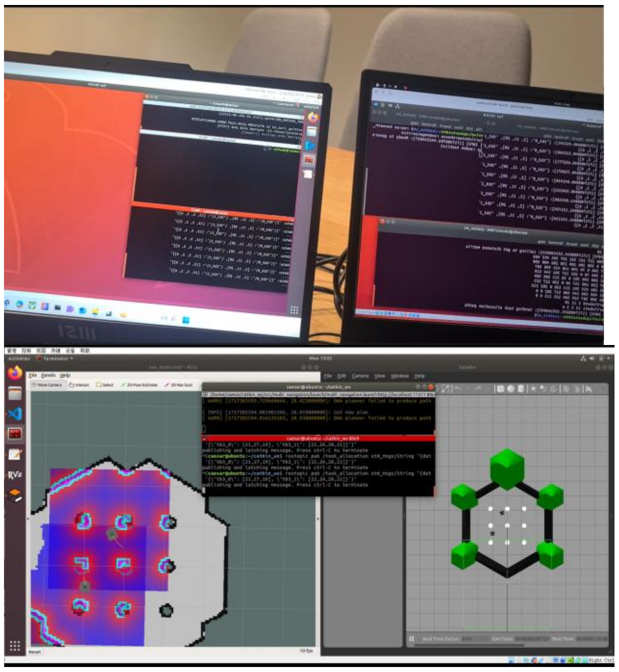
Table 1

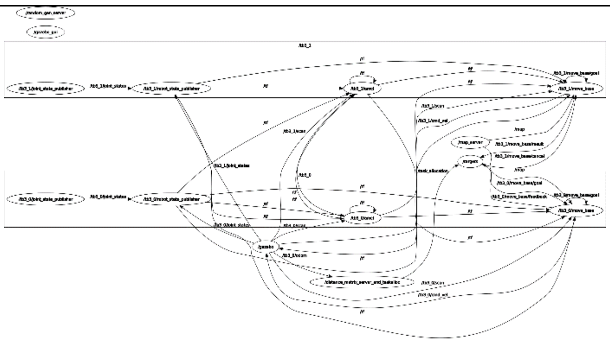
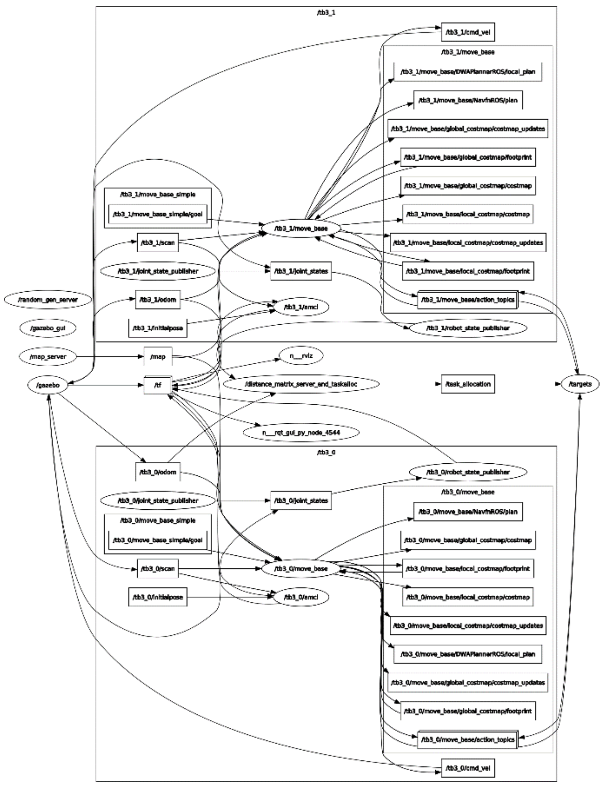
System Integration Test Plan

The system integration tests included several critical steps to verify the coordination and performance of the path planning, SLAM, and task allocation subsystems within the multi-robot system.

Task	State	Evidence
Test 1: TurtleBot Integration Test Control Requirement: The test aims to ensure that the TurtleBot can accurately follow movement commands and that the control algorithm package is effective in real-world scenarios. This involves two primary aspects: the TurtleBot's ability to move as commanded and the control algorithm's ability to accurately guide the TurtleBot along a planned path, including navigating around obstacles. **Test Procedure**: ● **Movement Capability Test:** ■ Send movement commands to the TurtleBot via the control repository, including forward, backward, and turning motions. ■ Monitor the TurtleBot's response to these commands and its execution of the corresponding movements. ● **Control Algorithm Package Validation:** ■ Utilize functions within the control algorithm package to generate path planning and movement instructions. ■ Apply these instructions to test if the TurtleBot can move accurately along the path planned by the algorithm package. ■ Observe the TurtleBot's response to environmental factors, such as obstacle avoidance and path adjustment behaviors, to assess the practical effectiveness of the control algorithm. **Evaluation Criteria**: **Movement Response:** The TurtleBot should execute movements accurately and precisely according to the issued commands. **Effectiveness of the Control Algorithm:** The TurtleBot should be able to navigate according to the path planned by the control algorithm package, demonstrating path planning and obstacle avoidance capabilities.	Pass	 **Photo:** The photo above shows control TurtleBot movement **Video recording:** Please watch "Control TurtleBot movement. mp4" **Explanation:** We have successfully connected the TurtleBot3 and are able to use the control package to ensure that the TurtleBot3 can move along the trajectory. The control principle is Dynamixel control, which communicates and controls the TurtleBot3 through the 'cmd_vel' topic.

Test 2: Test SLAM and path planning **Requirement:** The test focuses on verifying the system's ability to generate a map using SLAM and then utilize this map for path planning. The path planner should be able to take the generated map and use it to formulate a path from a specified start position to a target position. **Test Procedure:** ● SLAM Simulation: ■ Initiate the SLAM process to explore the environment and generate a detailed map. This may involve the robot moving through a test area or a simulated environment, depending on the testing setup. ■ Ensure the map includes all necessary details for path planning, such as obstacles, open spaces, and potentially hazardous areas. ● Path Planning Implementation: ■ Input the start and target positions into the path planning algorithm. ■ Use the map generated from the SLAM process as the basis for path planning. ■ Execute the path planning algorithm to calculate a viable path from the start to the target position, considering obstacles and optimal routing. **Evaluation Criteria:** **Map Reception:** The system should successfully receive and process the map generated from the SLAM procedure. **Path Generation:** A path should be successfully generated based on the SLAM-derived map and the specified start and target positions. The generated path should be practical, safe, and efficient, demonstrating the effectiveness of the path planning algorithm.	Pass	 **Photo:** The photo above shows navigate and generate SLAM maps **Video recording:** Please watch “SLAM mapping. mp4” **Explanation:** We found two ways for subsystems to communicate with each other. The first is for the path planning to obtain the dynamic map of SLAM by subscribing to the '/map' topic, which is published by gmapping. The second method is for SLAM to save the static map to a designated folder, then read and publish it to the '/map ' topic via map_server, which is then subscribed to by the path planning.

Test 3: Test Task Allocation and Path Planning Integration **Requirement:** The goal of this test is to ensure that the task allocation system can communicate with and exchange data with the path planning system. This interaction should be capable of generating information about the distance matrix and task allocation lists for each TurtleBot. Specific requirements include: **Communication between Task Allocation and Path Planning:** The task allocation system must be able to send a list of tasks to the path planning system and be capable of receiving feedback from the path planning system. **Generation of Distance Matrix and Task Lists:** Upon receiving the task list, the path planning system generates a corresponding distance matrix and feeds this information back to the task allocation system. The task allocation system then updates the task list for each TurtleBot based on the distance matrix. **Test Procedure:** ● Distance Matrix Generation and Sharing: ■ The path planning system initiates the process by generating a distance matrix between the tartgets and each robot. ■ This distance matrix is then shared with the task allocation system as the foundation for task allocation. ● Task Allocation and Task List Creation: ■ Upon receiving the distance matrix, the task allocation system uses it to allocate tasks among the available TurtleBots, considering factors such as proximity and task urgency. ■ A detailed task list for each TurtleBot is created and sent back to the path planning system. ● Path Planning for Each TurtleBot: ■ With the updated task lists, the path planning system calculates and generates optimal path plans for each TurtleBot, ensuring each robot efficiently completes its assigned tasks. **Evaluation Criteria:** Success is measured by the path planning system's ability to receive the task list and generate efficient path plans for each TurtleBot, allowing for the completion of tasks in an optimized manner.	Pass	 **Photo:** The interface highlights different TurtleBots, each assigned specific tasks and routes **Video recording:** Please watch “multi_targets_navigation.mp4” and “navigation_task_list.mp4” **Explanation:** The video "multi_targets_navigation.mp4" shows the process where multiple robots, upon receiving their task lists from the task allocation system, simultaneously execute multi-target navigation tasks and complete them. The video provides a detailed demonstration of how each TurtleBot dynamically receives task lists and executes assigned tasks based on the distance matrix and task urgency. Each robot, following the instructions from the task allocation system and supported by the path planning system, calculates the optimal path and performs the navigation tasks.

Integration Test 4: Full Simulation Test **Requirement:** The objective of this test is to conduct a full simulation involving one iteration of the task list, where task allocation, path planning, and SLAM are integrated to produce an output for a set task list. The robots must successfully move to the correct locations assigned to them. This requires: Integration of SLAM, Task Allocation, and Path Planning: All subsystems must work together seamlessly to utilize the map generated by SLAM for task allocation and path planning. Successful Navigation to Assigned Locations: Each robot must accurately navigate to its designated locations as per the task list, demonstrating the effectiveness of the integrated system. **Test Procedure:** **SLAM Map Generation:** Begin the simulation with SLAM to create a detailed map of the environment. This map is essential for accurate task allocation and path planning. **Task Allocation:** Based on the generated map and the given task list, allocate tasks to each robot, ensuring the tasks are feasible within the mapped environment. **Path Planning:** Generate path plans for each robot based on the task allocations. These plans should take into consideration the layout of the environment as defined by the SLAM map to ensure efficient and obstacle-free navigation. **Execute Movements:** The path plans are sent to the robots. Each robot follows its path plan to move to the correct locations assigned to it. **Evaluation Criterion:** The criterion for a successful test is that each TurtleBot moves to all locations specified on the task list successfully.	Pass	 /tb3_1 /tb3_1/cmd_vel /tb3_1/move_base /tb3_1/move_base/DWAPlannerROS/local_plan /tb3_1/move_base/NavfnROS/plan /tb3_1/move_base/global_costmap/costmap_updates /tb3_1/move_base/global_costmap/footprint /tb3_1/move_base/global_costmap/costmap /tb3_1/move_base/local_costmap/costmap /tb3_1/move_base/local_costmap/costmap_updates /tb3_1/move_base/local_costmap/footprint /tb3_1/move_base/action_topics /tb3_1/move_base_simple /tb3_1/move_base_simple/goal /tb3_1/scan /tb3_1/joint_state_publisher /tb3_1/joint_states /random_gen_server /tb3_1/odom /tb3_1/amcl /gazebo_gui /tb3_1/initialpose /tb3_1/robot_state_publisher /map_server /map n__rviz /gazebo /tf /distance_matrix_server_and_taskalloc /task_allocation /targets n__rqt_gui_py_node_4544 /tb3_0 /tb3_0/odom /tb3_0/robot_state_publisher /tb3_0/joint_state_publisher /tb3_0/joint_states /tb3_0/move_base /tb3_0/move_base_simple /tb3_0/move_base_simple/goal /tb3_0/move_base/NavfnROS/plan /tb3_0/move_base/global_costmap/costmap /tb3_0/scan /tb3_0/move_base/local_costmap/footprint /tb3_0/initialpose /tb3_0/amcl /tb3_0/move_base/local_costmap/costmap /tb3_0/move_base/local_costmap/costmap_updates /tb3_0/move_base/DWAPlannerROS/local_plan /tb3_0/move_base/global_costmap/costmap_updates /tb3_0/move_base/global_costmap/footprint /tb3_0/move_base/action_topics /tb3_0/cmd_vel **Photo:** Include an image that visually represents the integration of the SLAM, task allocation, and path planning processes. **Explanation:** During this integration test, the subsystems for SLAM, task allocation, and path planning must work cohesively. The generated map from SLAM provides the necessary spatial information for task allocation. Each robot's path planning module uses this map to generate viable paths, ensuring each robot can navigate to its designated locations efficiently. The test validates the system's capability to perform complex tasks in a coordinated manner, simulating real-world scenarios where multiple robots need to operate simultaneously in a shared environment.

Test 5: Computational Load Test (Optional) **Requirement:** This test is designed to assess the laptop's capability to handle the computational demands of a comprehensive warehouse simulation integrating multiple subsystems. These include running various algorithms and simulations simultaneously, which are essential for the operation of a dynamic warehouse environment. The laptop used must be able to continuously run the entire simulation setup without crashing or experiencing significant performance degradation. Key requirements include: **System Stability:** The laptop must maintain operational stability throughout the test, without any system crashes or freezes. **Performance Maintenance:** The laptop should not suffer from significant performance issues that could impede the running of simulations or algorithms. This includes managing thermal loads to prevent throttling and ensuring sufficient memory and processing resources are available. **Test Procedure:** **Setup and Initialization:** Configure the laptop with the necessary software and simulations that represent the integrated warehouse system. This setup should include all subsystems and the corresponding algorithms they run. **Simultaneous Operation:** Start all subsystems simultaneously on the laptop. This operation should mimic the computational load expected during the peak operation of the warehouse simulation. **Monitoring:** Throughout the test, monitor the laptop's performance metrics, including CPU and GPU usage, memory utilization, and thermal statistics. Use appropriate tools to log these metrics for later analysis. **Evaluation Criterion:** The laptop must complete the test without crashing, experiencing system freezes, or significant performance throttling. Successfully running the simulation without interruption and maintaining responsive system behavior throughout the test period are key indicators of passing this test.	No pass	**Explanation:** After our evaluation, it is temporarily impossible to run all the systems on a single laptop because our virtual machine may crash. Although it is possible to communicate using a server on different laptops within the same local area network, we no longer have time for this. Therefore, we can only give up this optional test.

Table 2

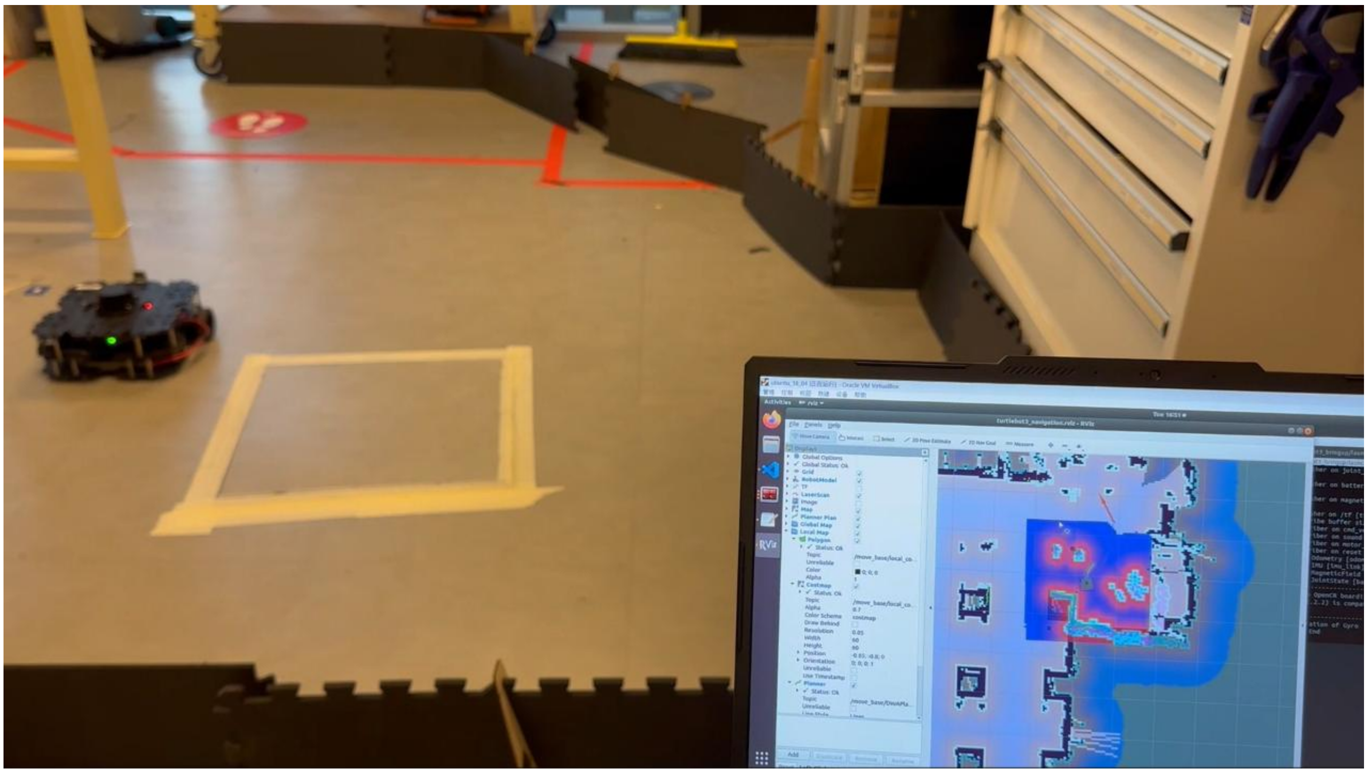
Other Contributions

Throughout the project, I also contributed significantly in other areas, including:

1. **TurtleBot Integration Test Control:**
   - **Objective:** To verify the TurtleBot's movement capabilities and the effectiveness of the control algorithm package in real-world scenarios.
   - **Activities:**
     - Sent movement commands to the TurtleBot and monitored its response.
     - Utilized functions within the control algorithm package to generate path planning and movement instructions.

- Ensured the TurtleBot accurately followed the planned paths and demonstrated effective obstacle avoidance.

- **Outcome:** Successfully connected and controlled TurtleBot3, ensuring it could move along the trajectory as planned. This included using Dynamixel control via the 'cmd_vel' topic for communication and control.
- **Evidence:**
  - **Photo:** Image showing TurtleBot3 successfully navigating a path.
  - **Video:** " Control_Pathplanning.mp4" demonstrating the TurtleBot following the planned path.
  - **Description:** Successfully executed commands and observed the TurtleBot navigating the planned path, avoiding obstacles, and responding accurately to control inputs.



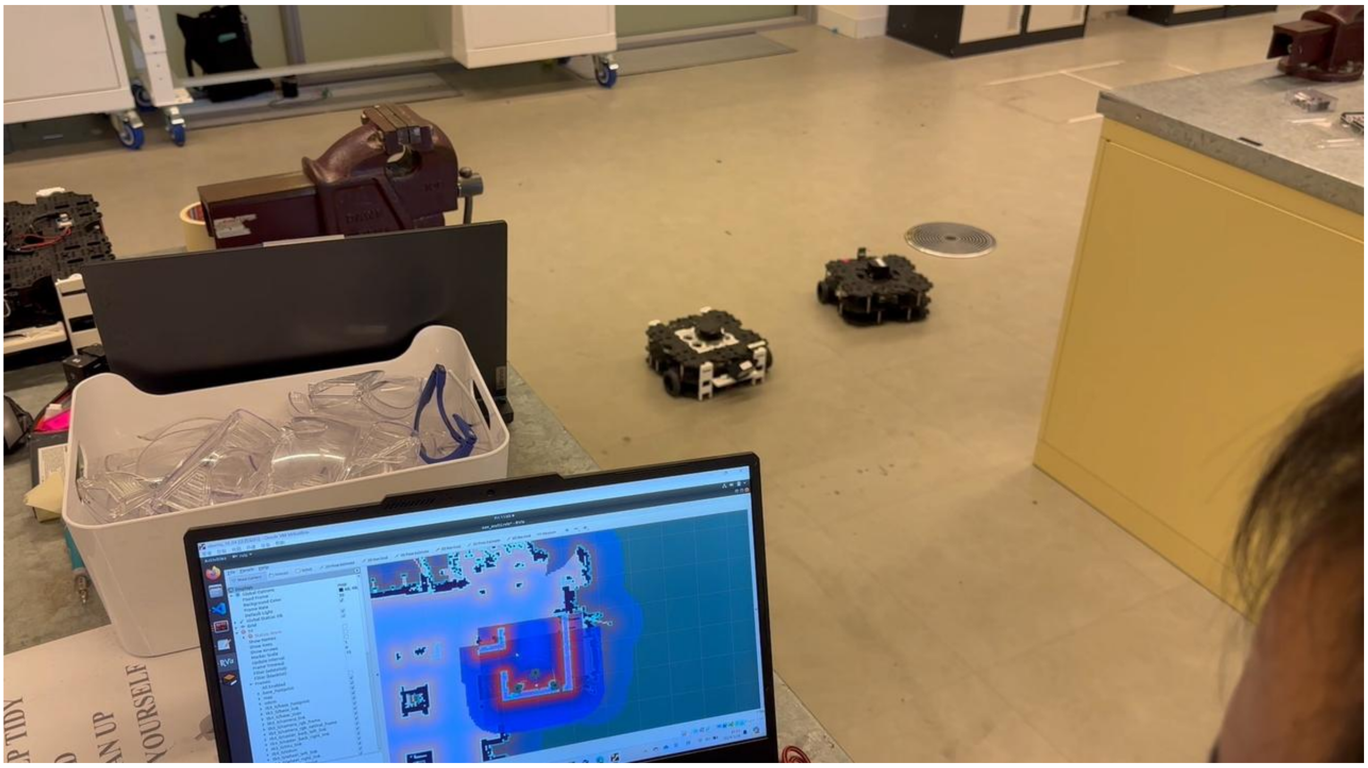
2. **Team Communication and Collaboration:**

- **Objective:** To foster effective communication and collaboration within the project team, ensuring smooth project progress and integration.
- **Activities:**
  - Actively participated in team meetings and discussions.
  - Assisted team members with technical issues and provided support for subsystem integration.
  - Coordinated efforts to ensure all team members were aligned with project goals and timelines.

- **Outcome:** Maintained a high level of teamwork and mutual support, contributing to the overall efficiency and success of the project.

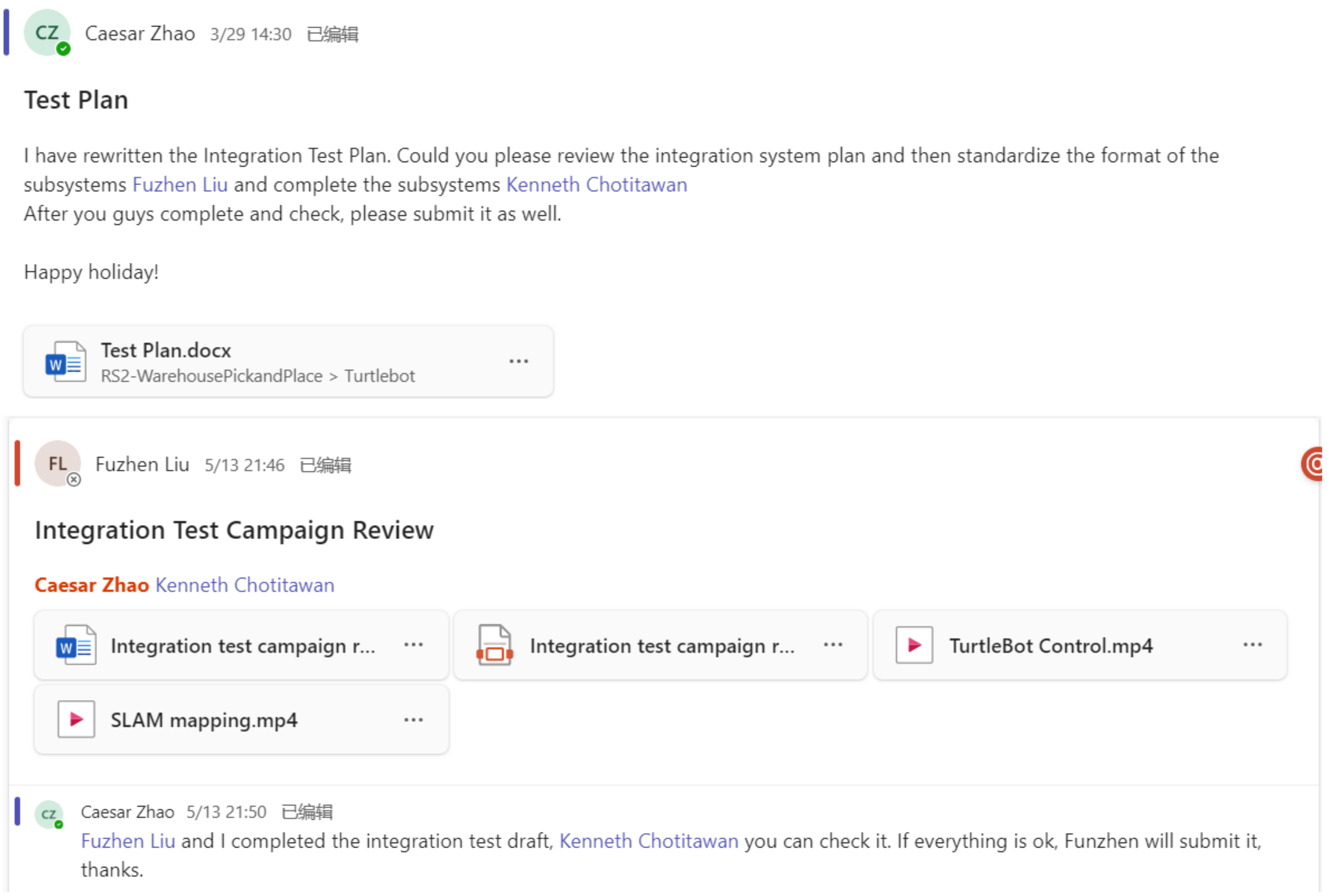
3. **Multi-Robot Connection:**

- **Objective:** To establish and verify the connection and communication between multiple robots within the system.
- **Activities:**
  - Configured network settings and communication protocols for multiple robots.
  - Tested and validated the connection between robots to ensure seamless data exchange and coordination.
- **Outcome:** Successfully completed the connection of multiple robots, enabling effective multi-robot coordination and task execution.
- **Evidence:**
  - **Network Configuration Files:** Screenshots showing network settings for multiple robots.
  - **Test Video:** Video recording successful connection tests between multiple robots. Please watch “Multi-robot_dynamic_obstacl_avoidance.MOV”, “Multi-robot_test.MOV” and “Multi-robot_connection.MOV”
  - **Description:** Configured and tested the network settings for multiple robots, ensuring seamless communication and coordination, which enabled effective multi-robot operations.



4. **Phase Reporting:**

- **Objective:** To document project progress, test results, and other relevant information at each project phase.
- **Activities:**
  - Prepared detailed reports summarizing the objectives, procedures, results, and conclusions of each testing phase.
  - Compiled and presented findings to the project stakeholders, ensuring transparency and accountability.
- **Outcome:** Provided comprehensive documentation and progress tracking, facilitating informed decision-making and project management.
- **Evidence:**
  - **Reports and Plans:** Submitted reports and plans used for project.
  - **Video Materials:** Links or summaries of recorded videos, showcasing key project activities and outcomes. It is shown in Figure



Caesar Zhao 3/29 14:30 已编辑

Test Plan

I have rewritten the Integration Test Plan. Could you please review the integration system plan and then standardize the format of the subsystems Fuzhen Liu and complete the subsystems Kenneth Chotitawan
After you guys complete and check, please submit it as well.

Happy holiday!

Test Plan.docx
RS2-WarehousePickandPlace > Turtlebot

Fuzhen Liu 5/13 21:46 已编辑

Integration Test Campaign Review

Caesar Zhao Kenneth Chotitawan

Integration test campaign r...
Integration test campaign r...
TurtleBot Control.mp4
SLAM mapping.mp4

Caesar Zhao 5/13 21:50 已编辑
Fuzhen Liu and I completed the integration test draft, Kenneth Chotitawan you can check it. If everything is ok, Funzhen will submit it, thanks.

Figure 2

These additional contributions demonstrate my ability to handle a wide range of tasks and responsibilities, showcasing my technical skills, teamwork, and dedication to the project's success. They also highlight my commitment to ensuring thorough testing, effective communication, and robust system integration.

Subsystem Function Description

Multi-Robot Path Planning Subsystem

The multi-robot path planning subsystem is designed to provide efficient and collision-free navigation for multiple robots operating within a warehouse environment (Madridano et al., 2021). This subsystem integrates global and local path planning algorithms to ensure optimal pathfinding and real-time obstacle avoidance. Below is a detailed description of its components and functionalities.

Repository Overview

The complete source code, configuration files, and documentation for the multi-robot path planning subsystem are available in the GitHub repository https://github.com/caesar1457/Robotics-Studio-2.git. The repository is structured to provide clear organization and easy access to all necessary components for setting up and running the system.

Key Components

1. **Namespace Configuration**:
   - The subsystem supports running multiple robots in different namespaces to avoid conflicts and ensure smooth operation.
   - To bring up robots in different namespaces, use the following commands:

```
ROS_NAMESPACE=tb3_0 roslaunch turtlebot3_bringup turtlebot3_robot.launch
ROS_NAMESPACE=tb3_1 roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

2. **Launching the Main Navigation System**:
   - The main navigation system is launched using the ‘multi_navigation.launch’ file. This file configures the necessary parameters and nodes for coordinating multiple robots.To bring up robots in different namespaces, use the following commands:
   - Before launching, set the TurtleBot3 model environment variable:

```
export TURTLEBOT3_MODEL=waffle_pi
```

   - Then, launch the main navigation system:

```
roslaunch multi_navigation multi_navigation.launch
```

3. **Task Allocation and Multi-target Navigation**:
   - The subsystem supports dynamic task allocation, allowing robots to be assigned multiple targets efficiently.
   - An example command to send multi-targets to the robots is:

```
rostopic pub /task_allocation std_msgs/String "{data: '{"tb3_0": [19,21,27,19], "tb3_1": [22,26,28,21]}'}"
```

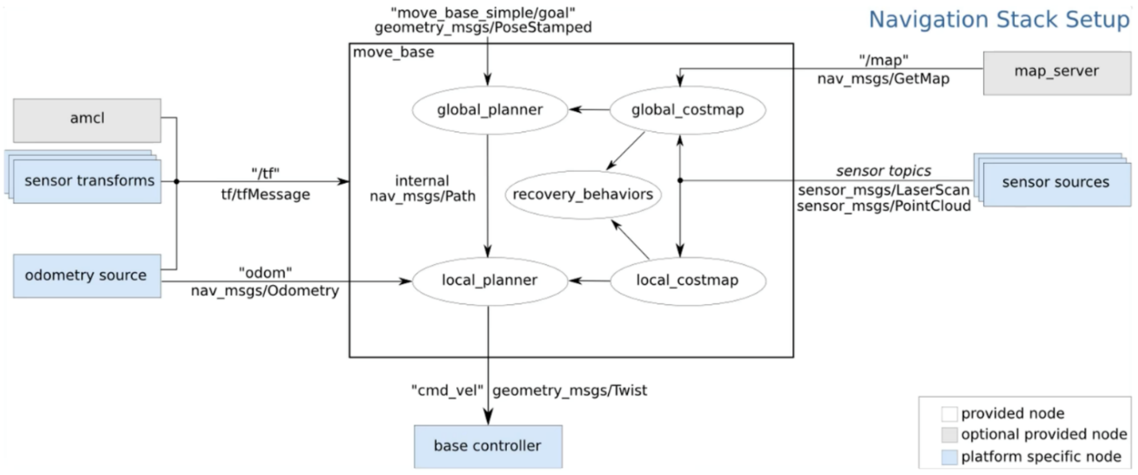
- This command publishes a message to the ‘/task_allocation’ topic, specifying the target points for each robot.

Summary

The multi-robot path planning subsystem in the Robotics Studio 2 project demonstrates a robust approach to managing multiple robots within a shared environment. The use of namespaces, combined with efficient path planning and task allocation strategies, ensures smooth and effective navigation. All related code and documentation are accessible through the project's GitHub repository, providing a comprehensive resource for understanding and deploying the subsystem.

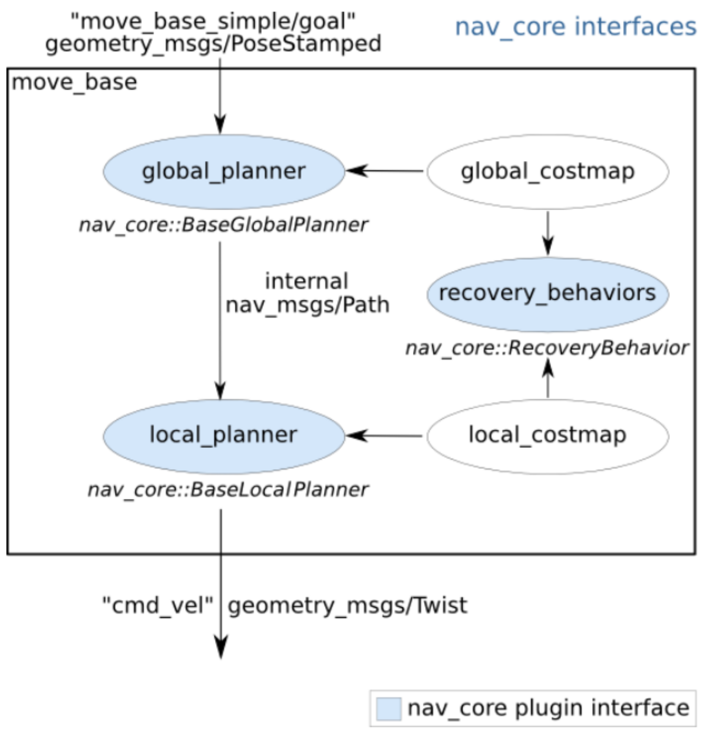
Navigation

- **Function:** The navigation component orchestrates the movement of the robots by integrating path planning, sensor data, and control signals. It ensures the robots can navigate through the environment while avoiding obstacles and reaching their targets efficiently (Lin et al., 2022).
- **Process:**
    - **Input**: Sensor data from laser scans, odometry, and transforms; goal positions.
    - **Components**:
        - **move_base**: Integrates global and local planners to generate feasible paths (Marder-Eppstein, n.d.).
        - **Global Planner**: Computes an initial path to the goal using a global path planning algorithm (e.g., Dijkstra or A*) (Red Blob Games, n.d.).
        - **Local Planner**: Adjusts the path dynamically to avoid obstacles in real-time using a local path planning algorithm (e.g., DWA) (Red Blob Games, n.d.)
        - **Costmaps**: Represent the environment's layout and obstacles, used by planners to calculate paths (Marder-Eppstein, n.d.).
    - **Output**: Navigation commands (`cmd_vel`) to control the robot's movement.
        - Implementation: The navigation stack is implemented using ROS with the move_base package, integrating sensor data through sensor_msgs and transforms (tf), and utilizing planners to navigate the environment (Binder et al., n.d.).



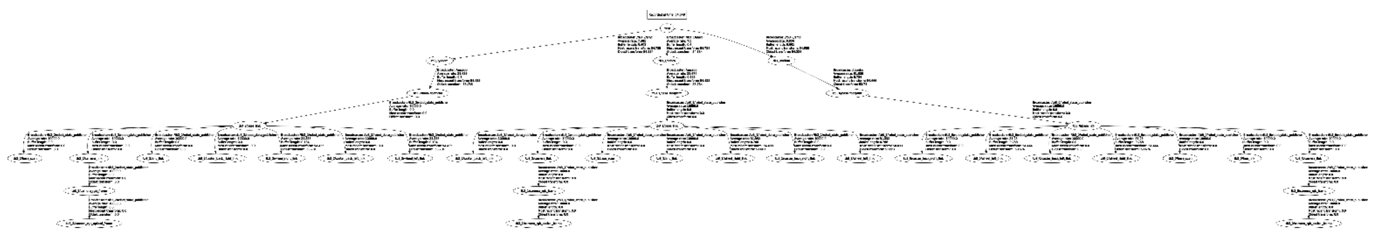
Recovery Behavior

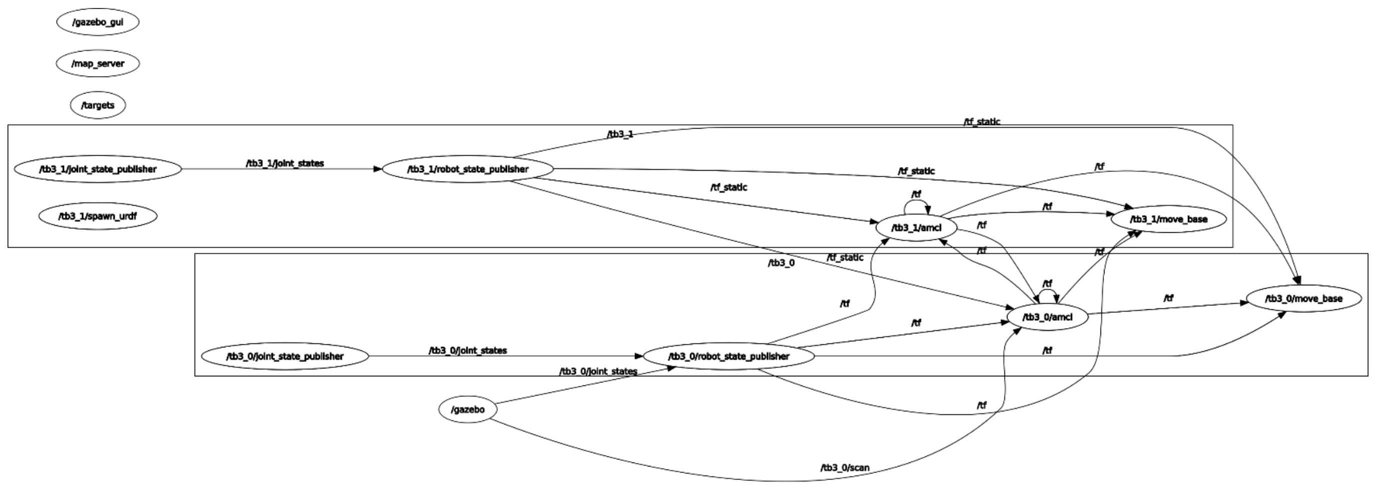
- **Function:** Recovery behaviors in the navigation stack are essential for enabling the robot to handle situations where it becomes stuck or encounters issues that prevent it from following the planned path. These behaviors help in resetting the robot's position or taking corrective actions to ensure it can continue navigating towards its goal (Binder et al., n.d.; Marder-Eppstein, n.d.).
- **Process:**
  - **Input:** Information about the robot's current state, sensor data, and feedback from the navigation stack indicating a failure or obstacle.
  - **Components:**
    - **Recovery Behaviors:**
      - **Clear Costmap:** This behavior clears the local and/or global costmaps to remove obstacles that might be no longer relevant but are still considered by the robot.
      - **Rotate Recovery:** Involves the robot performing in-place rotations to find a new path or to free itself from tight spots.
      - **Custom Recovery Behaviors:** Users can implement custom recovery actions suited to specific environments or robot capabilities.
  - **Output:** Corrective actions that modify the robot's position or path, enabling it to resume normal navigation.
    - **Implementation:** Recovery behaviors are implemented as plugins in the ROS navigation stack, allowing for flexibility and customization. These behaviors can be triggered automatically based on specific conditions or manually through commands (Marder-Eppstein, n.d.; Robotics Stack Exchange, n.d.).


"move_base_simple/goal"
geometry_msgs/PoseStamped
nav_core interfaces
move_base
global_planner
global_costmap
nav_core::BaseGlobalPlanner
internal
nav_msgs/Path
recovery_behaviors
nav_core::RecoveryBehavior
local_planner
local_costmap
nav_core::BaseLocalPlanner
"cmd_vel" geometry_msgs/Twist
nav_core plugin interface

Coordinate Frames and Namespace Management

- **Function:** The coordinate frames and namespace management component ensures the spatial relationships between different coordinate frames are maintained accurately and that multiple robots can operate in the same environment without conflicts (Robotics Stack Exchange, n.d.).
- **Process:**
  - **Input**: Transforms from various sensors and odometry sources.
  - **Components**:
    - **TF Tree**: Manages and updates the coordinate frames for the robot and its components (Marder-Eppstein, n.d.).
    - **Namespace**: Separates the ROS topics, parameters, and nodes for each robot to prevent data collisions and ensure independent operation.
  - **Output**: Consistent and accurate frame transformations, isolated topic namespaces for each robot.
    - **Implementation:** The TF tree is set up using the ‘tf’ package in ROS, with each robot’s namespace defined to ensure unique and conflict-free communication within a multi-robot system. The setup is visualized and monitored using tools like ‘rqt_tf_tree’.





Global Path Planning: Dijkstra Algorithm

- **Function:** The global path planning component uses the Dijkstra algorithm to calculate the shortest path from the robot's starting position to its destination. This algorithm ensures that the path is optimal in terms of distance, taking into account the static layout of the warehouse (Red Blob Games, n.d.).
- **Process:**
  - **Input:** The static map of the warehouse, the robot's current position, and the target position.
  - **Algorithm:** The Dijkstra algorithm processes the map as a weighted graph, where nodes represent locations and edges represent possible paths with associated costs (distances).
  - **Output:** A series of waypoints that form the shortest path from the start to the destination.
- **Implementation:** The algorithm is implemented in Python and visualized using Rviz in ROS. The paths generated are sent to the robots as navigation goals (Marder-Eppstein, n.d.).

Global Path Planning: A* Algorithm

- **Function:** Initially planned, the A* algorithm was considered for global path planning due to its efficiency and heuristic approach, which accelerates the search process by estimating the cost to the goal (Red Blob Games, n.d.).
- **Process:**
  - **Input:** The static map of the warehouse, the robot's current position, and the target position.

    - **Algorithm:** The A* algorithm uses a heuristic to guide the search for the shortest path, combining the actual cost to reach a node and an estimated cost to the goal.
    - **Output:** A set of waypoints representing the optimal path.
- **Implementation:** Although A* was initially considered, Dijkstra was ultimately chosen for the final implementation due to specific project requirements (Marder-Eppstein, n.d.).

Local Path Planning: Dynamic Window Approach (DWA)

- **Function:** The local path planning component employs the Dynamic Window Approach (DWA) to handle real-time obstacle avoidance and path adjustments (Julian98, 2020).
- **Process:**
    - **Input:** Sensor data (e.g., from LIDAR or cameras), current robot velocity, and the waypoints from the global path planner.
    - **Algorithm:** DWA calculates the best velocity commands to ensure smooth and collision-free movement by considering the robot's dynamics and nearby obstacles.
    - **Output:** Velocity commands that the robot can execute to follow the path while avoiding obstacles.
- **Implementation:** DWA is implemented in Python and integrated with the ROS navigation stack to control the robot's movements (Marder-Eppstein, n.d.).

Local Path Planning: Artificial Potential Field (APF)

- **Function:** Initially planned, the Artificial Potential Field (APF) method was considered for local path planning due to its simplicity in modeling attractive and repulsive forces for navigation (Sheng et al., 2010; Gu et al., 2019; Zhang et al., 2021). Please watch "Original_APF.mp4" and "Improved_APF.mp4"
- **Process:**
    - **Input:** Current robot position, target position, and locations of obstacles.
    - **Algorithm:** The APF generates attractive forces towards the target and repulsive forces away from obstacles. In the improved version, the repulsive force function also considers the distance from the robot to the target, addressing the unreachable problem by modifying the repulsive force to be influenced by this distance (Sheng et al., 2010; Gu et al., 2019; Zhang et al., 2021).
    - **Output:** Path adjustments based on the combined potential field.
- **Implementation:** Though APF was explored, DWA was chosen for its more robust handling of dynamic environments and robot dynamics. The improved APF showed better handling of local minima and unreachable targets by incorporating the distance to the target in the repulsive force calculation (Du & Nan, 2016; Fan et al., 2020).
- **Improved APF**:

- The improved APF modifies the repulsive force function to also consider the distance from the robot to the target, thereby addressing the problem of local minima and making previously unreachable targets accessible. This method ensures a more efficient path planning by balancing the attractive and repulsive forces more effectively (Du & Nan, 2016; Fan et al., 2020).

$$\overrightarrow{F_{\text{rep}}} = \begin{cases} \sum_{\text{i}} \left(\overrightarrow{\text{F}_{\text{rep1,ı}}} + \overrightarrow{\text{F}_{\text{rep2,ı}}}\right) & if \;\; d_i \le r_{rep} \\ 0 & if \;\; d_i \; > r_{rep} \end{cases}$$

where:

$$\overrightarrow{F_{\text{rep1},\imath}} = k_{\text{rep}} \cdot \left(\frac{1}{d_i} - \frac{1}{r_{\text{rep}}}\right) \cdot \frac{1}{d_i^2} \cdot |\overrightarrow{d_{\text{goal}}}|^2 \cdot \overrightarrow{d_{\text{obs},\imath}}$$

$$\overrightarrow{F_{\text{rep2},\imath}} = k_{\text{rep}} \cdot \left(\frac{1}{d_i} - \frac{1}{r_{\text{rep}}}\right)^2 \cdot |\overrightarrow{d_{\text{goal}}}| \cdot \overrightarrow{d_{\text{goal}}}$$

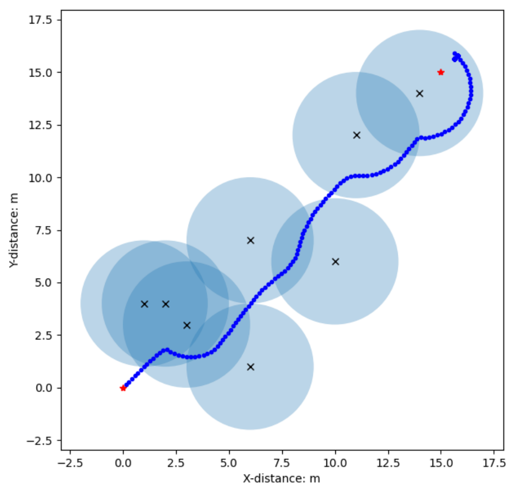


Figure 3 Original APF

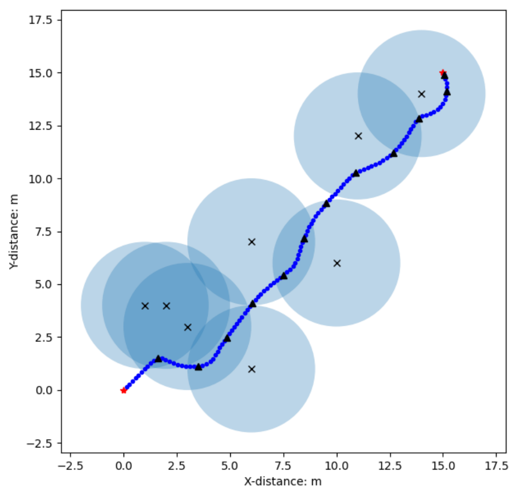


Figure 4 Improved APF

Soft Skills and Team Collaboration

Throughout the project, effective communication, reflection, peer collaboration, and professional conduct played vital roles in our success.

Good Communication with Clients

I actively communicated with my tutor and team members to ensure the project's smooth progress. Every week, I attended classes and lab sessions, discussing project details with the team. For instance

- **With Fuzhen**: We discussed system integration, ROS system details (RViz, Gazebo), and the use of TurtleBot3, including its connection and the integration of SLAM and path planning. We provided technical support to each other, enhancing our communication skills and teamwork.
- **With Ken**: We discussed the overall system flowchart, ROS system selection, and the interface between path planning and task allocation.

- **With Quang**: We focused on each project's stage, the challenges encountered, our proposed solutions, project expectations, and ways to improve communication and collaboration.

Challenges and Solutions

Throughout the project, I encountered numerous challenges that required overcoming fears and self-directed learning. These challenges included:

- Installing and using the ROS system.
- Using Python programming language.
- Implementing A*, APF, Dijkstra, and DWA algorithms.
- Utilizing and linking TurtleBot3.
- Using and editing Gazebo and RViz files, including launch files.
- Managing multiple namespaces.
- Ensuring a closed-loop TF tree.
- Linking single and multiple TurtleBot3 units.
- Effective team communication.
- Structuring the overall system.
- Designing the subsystem framework.
- Overcoming fear and frustration, staying motivated under pressure.
- Efficient self-learning.
- Effective English communication.
- Registering move_base plugins.
- Writing and subscribing to topics.

I extensively researched online resources and videos, gradually solving these technical issues through self-learning. When facing prolonged difficulties or negative emotions, I actively communicated with Quang and Fuzhen, exchanging opinions and solving problems. We also encouraged each other, progressively overcoming all challenges.

Reflections

During Design Review 3, I reflected on several key points:

- Reliability of preliminary research.
- Reliability of the overall framework and project goals.
- Reliability of subsystems.
- Allowing sufficient buffer time.
- Understanding the implementation process before starting the project.
- Having backup plans.
- Necessity of subsystem and integration system testing.

From the project, I learned the following key lessons and improvement measures:

- Ensure detailed research on the subsystem process during preliminary research, linking each stage until completion.
- Confirm the project's framework reliability and set achievable goals through discussions and research.
- Allocate one to two weeks as buffer time during project planning.
- Prepare alternative solutions early, not relying on a single plan.
- Capture critical stages during subsystem and integration system testing, avoiding redundancy and omissions.

Peer Assessments

Peer feedback highlighted my positive impact on team dynamics. I actively communicated, offering advice and comfort during setbacks, helping the team reorganize and regain focus.

- **Providing Constructive Feedback**: I offered thoughtful and constructive feedback during meetings, helping team members refine their ideas and approaches.
- **Encouraging Team Morale**: During setbacks, I provided comfort and encouragement, which helped maintain a positive and productive atmosphere.
- **Facilitating Problem-Solving**: By actively listening and offering solutions, I helped the team navigate through difficult issues and stay on track. This included debugging code issues together and recommending relevant books and videos for better understanding of subsystems.
- **Promoting Inclusivity**: I ensured everyone had a voice in discussions, fostering a collaborative and inclusive team environment.

Professional Conduct in Labs and Studios

In the lab, I demonstrated professional behavior by:

- **Handling Equipment Carefully**: I handled TurtleBot3 and other equipment with care, ensuring they were returned promptly after use and stored properly.
- **Maintaining Cleanliness and Order**: I avoided shouting, eating, or causing disruptions in the lab, maintaining a professional and respectful environment.
- **Respecting Lab Protocols**: I adhered to all lab protocols, including proper usage and storage of power supplies and other sensitive equipment.
- **Collaborative Etiquette**: I respected the workspace of my peers, ensuring a cooperative and harmonious working environment.
- **Safety Awareness**: I followed all safety guidelines and encouraged my team to do the same, prioritizing the well-being of everyone in the lab.

These efforts in maintaining professionalism and effective team collaboration significantly contributed to the project's success, ensuring efficient problem-solving, effective communication, and a positive working environment throughout the project lifecycle.

Project and Time Management

- Created detailed project plans and timelines, ensuring tasks were completed on schedule.
- Regularly reviewed and adjusted plans to accommodate unexpected challenges.
- Prioritized tasks effectively, focusing on critical milestones.
- Maintained clear and organized documentation for all project activities.

These soft skills and team collaboration efforts significantly contributed to the project's success, ensuring efficient problem-solving, effective communication, and professional conduct throughout the project lifecycle.

Conclusion

The multi-robot path planning project successfully achieved its objectives of designing and implementing a robust system capable of efficient and collision-free navigation in a dynamic warehouse environment. Through the integration of advanced path planning algorithms and effective coordination mechanisms, the system demonstrated its ability to handle complex tasks involving multiple robots.

Key findings and outcomes

- **Effective Global Path Planning:**
  - The Dijkstra algorithm was effectively utilized for global path planning, ensuring that the shortest and most efficient paths were calculated for robots to navigate from their starting points to their destinations. This algorithm's accuracy and reliability were confirmed through extensive testing and real-world simulations.
- **Dynamic Local Path Planning:**
  - The Dynamic Window Approach (DWA) provided robust real-time obstacle avoidance and path adjustment capabilities. The integration of DWA with the ROS navigation stack enabled the robots to navigate safely through dynamic environments, avoiding both static and moving obstacles.
- **Initial Considerations and Adaptations:**
  - The project initially explored the use of the A* algorithm for global path planning and the Artificial Potential Field (APF) method for local path planning. Although these methods were not used in the final implementation, the insights gained from their exploration contributed to the project's overall success.
- **Seamless Coordination and Communication:**
  - The task assignment module efficiently distributed tasks among the robots, while the SLAM module provided real-time map updates and localization. The inter-robot communication mechanisms ensured that robots could share positional and path information, preventing collisions and optimizing overall system efficiency.
- **Comprehensive Testing and Validation:**

    - Extensive testing validated the system's ability to generate and follow paths, avoid dynamic obstacles, and coordinate multiple robots. The successful execution of these tests demonstrated the system's reliability and robustness in handling real-world scenarios.

Suggestions for future work

- **Enhanced Algorithm Integration:**
    - Extensive testing validated the system's ability to generate and follow paths, avoid dynamic obstacles, and coordinate multiple robots. The successful execution of these tests demonstrated the system's reliability and robustness in handling real-world scenarios.
- **Advanced Obstacle Avoidance:**
    - Investigating and implementing more sophisticated local path planning methods, such as machine learning-based approaches, could enhance the system's ability to handle highly dynamic and unpredictable environments.
- **Scalability Improvements:**
    - Enhancing the system's scalability to manage a larger fleet of robots would be beneficial for larger warehouse operations. This could involve optimizing communication protocols and task assignment strategies.
- **Real-World Deployment:**
    - Conducting real-world deployment and testing in an actual warehouse setting would provide valuable insights and help refine the system to meet practical operational challenges.

The multi-robot path planning system developed in this project has demonstrated its capability to provide efficient, safe, and reliable navigation solutions in a warehouse environment. The successful integration of Dijkstra and DWA algorithms, coupled with effective coordination and communication mechanisms, has ensured the system's robustness and practicality. The lessons learned and the outcomes achieved lay a strong foundation for further advancements and real-world applications in multi-robot systems.

References

Binder, B., Bader, M., & Beck, F. (n.d.). *tuw_multi_robot*. ROS Wiki. Retrieved from https://wiki.ros.org/tuw_multi_robot

Chen, J., Qin, X., Li, X., Zhou, Y., & Bao, B. (2020). Multi-robot collaborative obstacle avoidance based on artificial potential field method. *Computer Science*, *47*(11), 522-533. https://doi.org/10.11896/jsjkx.190900026

Du, Y., & Nan, Y. (2016). Research of Robot Path Planning Based on Improved Artificial Potential Field. . https://doi.org/10.2991/AMEII-16.2016.195.

Fan, X., Guo, Y., Liu, H., Wei, B., & Lyu, W. (2020). Improved Artificial Potential Field Method Applied for AUV Path Planning. *Mathematical Problems in Engineering*. https://doi.org/10.1155/2020/6523158 .

Gu, X., Han, M., Zhang, W., Xue, G., Zhang, G., & Han, Y. (2019). Intelligent Vehicle Path Planning Based on Improved Artificial Potential Field Algorithm. *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*, 104-109. https://doi.org/10.1109/HPBDIS.2019.8735451.

Julian98. (2020, July 26). Move_base namespace issue with multi-robot simulation. *ROS Answers: Open Source Q&A Forum*. Retrieved from https://answers.ros.org/question/362623/move_base-namespace-issue-with-multi-robot-simulation/

Lin, S., Liu, A., Wang, J., & Kong, X. (2022). A review of path-planning approaches for multiple mobile robots. *Machines*, *10*(9), 773. https://doi.org/10.3390/machines10090773

Luo, Q., Wang, H. B., Cui, X. J., & Xu, H. (2018). Research on autonomous navigation system of warehousing mobile robot based on improved artificial potential field method in dynamic environment. *Appl Res Comput*.

Madridano, Á., Al-Kaff, A., Martín, D., & De La Escalera, A. (2021). Trajectory planning for multi-robot systems: Methods and applications. *Expert Systems with Applications*, *173*, 114660. https://doi.org/10.1016/j.eswa.2021.114660

Marder-Eppstein, E. (n.d.). *move_base*. ROS Wiki. Retrieved from https://wiki.ros.org/move_base

Marder-Eppstein, E. (n.d.). nav_core. ROS Wiki. Retrieved from https://wiki.ros.org/nav_core

Qixin, C., Yanwen, H., & Jingliang, Z. (2006). An evolutionary artificial potential field algorithm for dynamic path planning of mobile robots. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3331-3336). https://doi.org/10.1109/IROS.2006.282551

Red Blob Games. (n.d.). *Pathfinding and search algorithms*. Retrieved from https://www.redblobgames.com/

Robotics Stack Exchange. (n.d.). Navigation - move_base with multiple robots running gmapping SLAM on TurtleBot3. Retrieved from https://robotics.stackexchange.com/questions/11634/navigation-move-base-with-multiple-robots-running-gmapping-slam-on-turtlebot3

rrt_exploration. (n.d.). *ROS Wiki*. Retrieved from https://wiki.ros.org/rrt_exploration

Sheng, J., He, G., Guo, W., & Li, J. (2010). An Improved Artificial Potential Field Algorithm for Virtual Human Path Planning. , 592-601. https://doi.org/10.1007/978-3-642-14533-9_60.

Yang, L., Li, P., Qian, S., Quan, H., Miao, J., Liu, M., ... & Memetimin, E. (2023). Path planning technique for mobile robots: A review. *Machines*, *11*(10), 980.

Zhao, J., & Zhu, Y. (2011). Mobile robot path planning based on improved artificial potential field method. *Journal of Harbin Institute of Technology, 81*-88. Retrieved from https://www.researchgate.net/publication/289893702_Mobile_robot_path_planning_based_on_improved_artificial_potential_field_method

Zhang, H., Li, M., & Wu, Z. (2021). Path Planning based on Improved Artificial Potential Field Method. *2021 33rd Chinese Control and Decision Conference (CCDC)*, 4922-4925. https://doi.org/10.1109/CCDC52312.2021.9602174.

Appendix

1. https://github.com/caesar1457/Robotics-Studio-2.git
   - **Description:** This repository contains all the code related to the Robotics Studio 2 project. It includes the source code, configuration files, and documentation for setting up and running the multi-robot path planning system. The repository is structured to provide a clear organization of the project's components, making it easy to navigate and understand the implementation details.
   - **Video Link**: https://github.com/caesar1457/Robotics-Studio-2.git
2. Trailer.mp4

   - **Description:** This video showcases the key achievements of the Robotics Studio 2 project, divided into four sections:
     - **A Visualized Search***: Demonstrates the A* algorithm in path planning, showing the optimal pathfinding process.
     - **Multi-Robot Dynamic Obstacle Avoidance**: Highlights real-time dynamic obstacle avoidance by multiple robots.
     - **Multi-Goal Navigation Allocation**: Displays efficient target allocation and navigation for multiple robots to various goals.
     - **Multi-Robot Connectivity and Path Planning**: Showcases coordinated path planning and communication between multiple robots.
   - **Video Link**: Trailer.mp4
3. Astar.mp4
   - **Description:** This video demonstrates the process and results of path planning using the A* algorithm. The robot navigates the simulated environment following paths generated by the A* algorithm, verifying the algorithm's effectiveness and efficiency.
   - **Video Link**: Astar.mp4
4. Control_Pathplanning.mp4
   - **Description:** This video showcases the integration test of the path planning and control systems. The robot navigates according to the planned paths, demonstrating the overall coordination and control effectiveness of the system.
   - **Video Link**: Control_Pathplanning.mp4
5. multi_navigation.mp4
   - **Description:** This video displays the navigation process of multiple robots within the same environment. It verifies the coordination and path planning capabilities of the multi-robot system, ensuring that each robot can effectively avoid obstacles and complete their respective tasks.
   - **Video Link**: multi_navigation.mp4
6. multi_robot_navi.mp4
   - **Description:** This video records the coordination of multiple robots executing navigation tasks in a complex environment. It highlights the communication and collaboration between robots to avoid collisions and optimize overall path efficiency.
   - **Video Link**: multi_robot_navi.mp4
7. SLAM_mapping.mp4
   - **Description:** This video demonstrates the process of the SLAM system generating a map of the environment and how the path planning system utilizes these maps for path planning. It verifies the integration of the SLAM and path planning systems.
   - **Video Link**: SLAM mapping.mp4
8. TurtleBot_Control.mp4
   - **Description:** This video shows the control test of the TurtleBot in a test environment. The TurtleBot receives and executes movement commands, demonstrating its movement capabilities and the effectiveness of the control algorithm.
   - **Video Link**: TurtleBot Control.mp4

9. Multi-robot_connection_test.mov
    - **Description:** This video demonstrates the connection test of multiple robots in the system. It shows how the robots are interconnected and communicate with each other to coordinate their actions.
    - **Video Link**: Multi-robot_connection.MOV
10. Multi-robot_dynamic_test.mov
    - **Description:** This video shows the dynamic obstacle avoidance test of multiple robots in a simulated environment. It highlights the robots' ability to dynamically navigate and avoid obstacles in real-time.
    - **Video Link**: Multi-robot_dynamic_obstacl_avoidance.MOV
11. Multi-robot_testing.mov
    - **Description:** This video presents the overall testing of multiple robots, including movement coordination and task execution. It illustrates the successful implementation and testing of the multi-robot system.
    - **Video Link**: Multi-robot_test.MOV
12. Original_APF.mp4
    - **Description:** This video demonstrates the implementation of the original Artificial Potential Field (APF) method for robot navigation. The robot navigates towards the target while avoiding obstacles using the standard APF algorithm.
    - **Video Link**: Original_APF.mp4
13. Improved_APF.mp4
    - **Description:** This video showcases the improved APF method, which enhances the repulsive force function by considering the distance from the robot to the target point. This modification aims to resolve the issue of unreachable targets. The robot demonstrates more effective navigation and obstacle avoidance compared to the original APF method.
    - **Video Link**: Improved_APF.mp4
14. multi_targets_navigation.mp4
    - **Description:** This video demonstrates the simultaneous navigation of multiple robots to multiple target points within a simulated environment. Each robot is assigned a set of unique target points and uses path planning algorithms to navigate to these points while avoiding obstacles and collisions with other robots. The video showcases the coordination and efficiency of multi-robot systems in complex navigation tasks.
    - **Video Link**: multi_targets_navigation.mp4
15. navigation_task_list.mp4
    - **Description:** This video illustrates the execution of a task list by a robot navigating within a simulated environment. The robot is tasked with visiting a sequence of designated points, showcasing its ability to prioritize tasks and efficiently plan its path. The video highlights the robot's capabilities in dynamic task management and obstacle avoidance using advanced path planning techniques.
    - **Video Link**: navigation_task_list.mp4