AURO REPORT

Examination Number: [Your Examination Number Here]
Module Code: COM00052H

I. Assessment Introduction

For this assessment, we've been tasked with programming autonomous mobile robots, specifically the TurtleBot3 Waffle Pi, to navigate through a simulated environment with obstacles, and retrieve items scattered around before returning them to the home zone.

II. Design

A. System Overview

In the design of my autonomous robot, I implemented a modular approach with the TurtleBot3, using the ROS2 framework. At the heart of the design is a finite state machine (FSM), which contains the robot's states - FORWARD, TURNING, COLLECTING, and RETURNING. I chose this FSM for its ability to provide structured yet flexible control, which is essential for navigation and adapting to dynamic task requirements. The keys to perception were the LiDAR and camera sensors, which are essential for effective operation. Their integration within ROS2 is indispensable for the task.

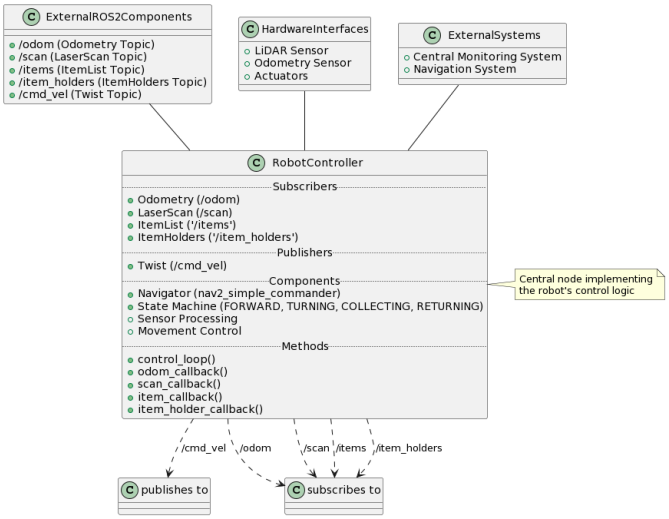

Fig. 1. High-Level System Design

B. Component Interaction

As shown in Fig. 1, the robot controller subscribes to several essential topics for efficient operation:

- **LaserScan**: Provides obstacle navigation data, taken from the LiDAR sensor, enabling the robot to detect and avoid obstacles.
- **ItemList**: Processes information from the camera to identify items within the robot's field of view, facilitating item identification and retrieval.
- **ItemHolders**: Keeps track of the items held by each robot, sourced from internal status monitoring. Helps in preventing task ovelap.
- **Odometry**: Provides data on the robot's position and orientation, taken from odometry sensors. It's essential for accurate navigation and precise maneuvering.

The data from these topics is parsed using the callback methods, a key process in the system's functionality.

III. Implementation

A. ROS Architecture

The implementation of my autonomous robot system is deeply rooted in the ROS 2 framework. It provides a modular and scalable architecture. This architecture is made from multiple ROS 2 nodes, each of which performs specific functions such as sensor data processing, state management, and motion control. The communication among these nodes is facilitated through topics, in which nodes publish and subscribe to messages.

B. Node Structure and Topic Communication

The system is structured around several key nodes. For example, the `robot_controller` node subscribes to the `/scan` topic for LiDAR data and the `/odom` topic for odometry information. It also publishes movement commands to the `/cmd_vel` topic.

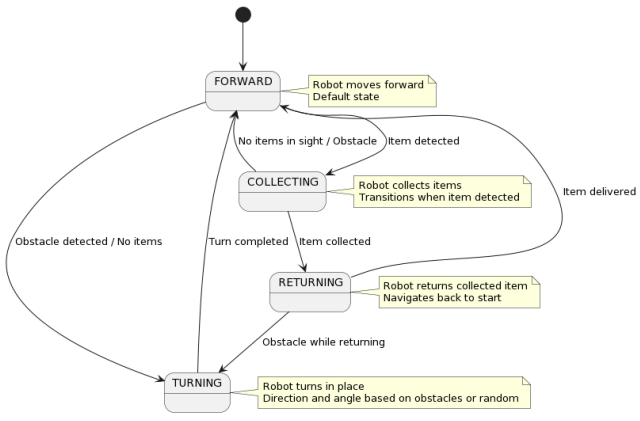

Fig. 2. State Machine Representing Autonomy

C. Autonomy Realization

Autonomy in the system is achieved through a well-defined finite state machine (FSM), as depicted in Fig. 2. The FSM has 4 states in which it can be: FORWARD, TURNING, COLLECTING, and RETURNING. It transitions based on sensor inputs and internal logic, also shown in Fig. 2.

D. Sensor Integration and Data Processing

Integrating sensors like LiDAR and cameras is key to the robot's perception capabilities. Using ROS 2, sensor data is processed in real-time, with custom callback functions handling the data for tasks like obstacle detection and item identification. Without this processed information, navigation decisions and effective task execution would be impossible.

E. Actuation and Movement Control

Movement control within the system is made through the generation of Twist messages. Those messages determine the robot's linear and angular velocities, and are based on the current state of the FSM and the processed sensor data. This shows a direct link between sensory input and actuator output.

F. Challenges and Solutions

During the early phases of the development process, I started with designing and implementing a system for a single robot. This approach, however, later caused issues whilst integrating the multi-robot solution. One such issue was all robots transitioning to the RETURNING state once one of them collected an item. This was due to the lack of optimisation for a multi-robot system. For instance, the `item_holder_callback` method was returning all the bots' information simultaneously, which caused the issue. The solution was to ensure each robot received and acted upon only the information relevant to itself. This change made the system effectively manage the state transitions for each robot independently and further align with the demands of a multi-robot system.

IV. Analysis

A. Experimental Approach

To evaluate the performance and efficiency of the system, I've taken a comprehensive experimental approach. This involved multiple strategies to gather both qualitative and quantitative data:

- **Data Logging**: Extensive data logging into the terminal was performed to capture real-time operational details of the system - This helped find bugs and solve issues.
- **Visual Inspections**: During simulations, visual inspections / Analysis were done to identify and fix issues in the system's behavior.
- **Simulation Analysis**: Detailed analysis of data logs from 10 different simulations, each lasting 5 minutes in real-time (approximately 1:30 minutes in simulation time in my environment). These simulations each used different item manager 'seeds' to assess the system's efficiency and adaptability under different conditions.

Repeated trials were also conducted to ensure the reliability and consistency of the system's performance across various scenarios.

B. Results

The results of the experiments are presented in Fig. 3 and Table I.

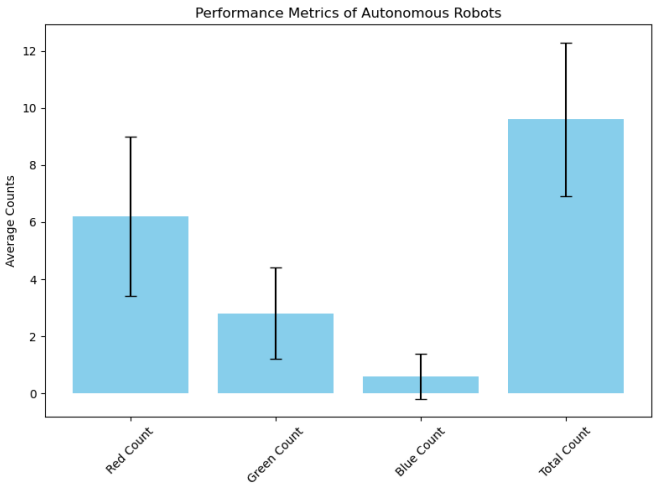

Fig. 3. Performance Analysis Graph

TABLE I
Performance Metrics

Metric	Average	Standard Deviation
Red Count	6.2	2.79
Green Count	2.8	1.60
Blue Count	0.6	0.80
Total Count	9.6	2.69

C. Results Analysis

Qualitatively, the system showed decent performance in item collection, with an average of approximately 10 items per simulation. Notably, red items were collected most frequently, as they were positioned closest to the home zone. Green items, typically located in the middle, were collected next in frequency, followed by blue items, which were the furthest away. Despite the robots being programmed to prioritize blue, then green, and then red items within their field of view (FOV) when collecting, this priority order was not reflected in the results.

Quantitatively, the reason behind this can be understood through visual inspections conducted during the simulations. While the robots initially prioritize blue items, they often collect green or red items en route to the blue ones, leading them to transition to the returning state prematurely. Additionally, on their way back to the home zone with a blue item, they frequently swap it for more readily accessible red items. This behavior is linked to the functionality of the item manager, though it is unclear whether this is a bug or an intentional

design choice. An alternative approach could have been to treat untargeted items as obstacles when the robot is in the returning state, which would prevent item swapping. However, this could significantly increase the time taken for the robot to return home, potentially decreasing the total count of collected items but possibly resulting in a higher count of blue and green items.

V. Evaluation

Discuss the strengths and weaknesses of your solution based on your design, implementation, and analysis. Address the transferability of your solution from simulation to reality.

VI. Safety and Ethics

Discuss safety implications and ethical considerations of autonomous robotic item retrieval in real-world scenarios. Reflect on how these relate to your solution.

VII. Conclusion

Summarize your findings and suggest future work or improvements.

References