AURO Report

1 Overview

This paper details the architecture and evaluation of an autonomous robotic system developed to retrieve and deposit items efficiently. To achieve this, a deliberative architecture was used, with reactive elements in navigation. The system developed is a decentralized homogenous multi-robot system and includes explicit communication to aid in planning. Inspiration was taken from microservice architecture, to maintain modularity and to facilitate efficient communication between robots. Microservice architecture is commonly used in the field of cloud robotics, and the Robot Operating System (ROS) is made up of distributed software nodes that are well suited to a microservice approach [1].

2 Architecture

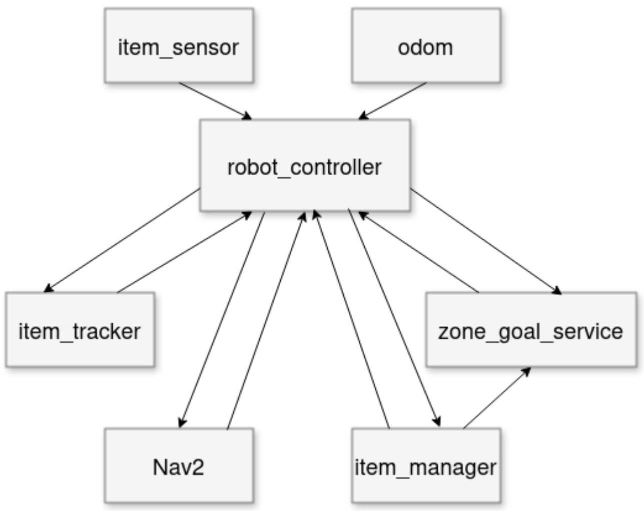

Figure 1: High-level architecture diagram

The robot_controller node implements the main control strategy using a finite state machine (FSM). Nav2 also implements some of the control strategy which is implemented through a behavior tree (BT). The robot_controller node receives information about visible items through the items topic from the item_sensor node. It estimates the 3D coordinates of the item, and uses the add_item service provided by the item_tracker node, which maintains a list of observed items. The robot_controller uses the get_item service provided by the item_tracker, to request the position of an item to collect. This information is used to set a Nav2 goal, which plans and follows a path to the item while avoiding obstacles. Nav2 uses many more nodes, such as planners, controllers, and navigators, but this is represented as one node in the diagram for simplicity.

When the item is reached, the pick_up_item service from the item_manager node is used to pick up the item, and then the set_zone_goal service from the zone_goal_service node is used to find the position of the optimal zone to return the item to. Nav2 is then used again to navigate to this zone, before dropping the item using the offload_item service from the item_manager node.

This modular approach to item tracking and zone assignment works well, and enables intelligent coordination, preventing robots from trying to collect the same tracked item, and allowing a robot to pursue an item that was added by another robot, reducing exploration time. This system also enables easy implementation of other tasks while changing minimal code, by changing logic in the item_tracker or the zone_goal_service.

3 Control

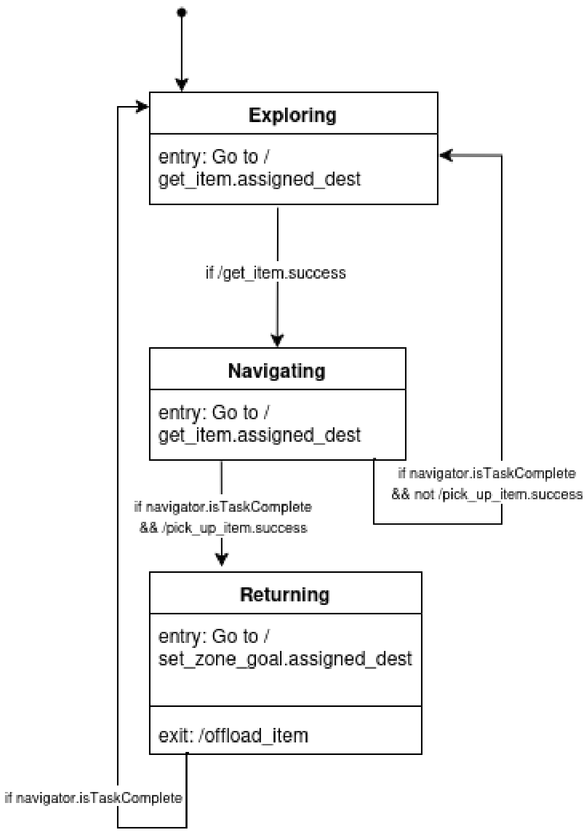
Exploring
entry: Go to / get_item.assigned_dest
if /get_item.success
Navigating
entry: Go to / get_item.assigned_dest
if navigator.isTaskComplete && not /pick_up_item.success
if navigator.isTaskComplete && /pick_up_item.success
Returning
entry: Go to / set_zone_goal.assigned_dest
exit: /offload_item
if navigator.isTaskComplete

Figure 2: robot_controller state machine

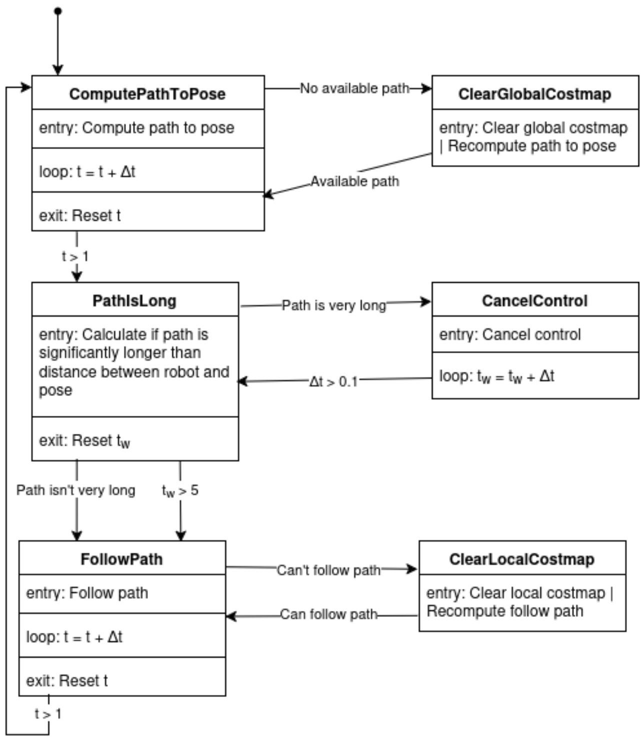

Figure 3: Nav2 behavior tree state machine

The FSM consists of 3 states: exploring, navigating, and returning. The exploration state takes place when there is not an item for a robot to collect, meaning the get_item service responds with a failure, but provides a location to navigate to (assigned_dest), so that the robot can hopefully see items on the way. The robot continually uses this service, and if it responds with success, it cancels any current navigation, begins navigating to the assigned item, and moves to the navigating state.

The navigating state does nothing until the navigation provided by Nav2 has finished, when it attempts to pick up an item using the pick_up_item service. If this fails, it goes back to the exploring state, but if it is successful, gets an assigned zone to drop the item at through the set_zone_goal service and moves to the returning state.

The returning state, waits until navigation has finished, then attempts to offload the item using the offload_item service. Then it moves to the exploring state. The simplicity of this FSM is a result of the microservice architecture, the main control logic is cleared due to offloading other computation to services.

The state machine provided here describes the navigation subtree of the behavior tree used, and omits the recovery subtree for clarity. This behavior tree is provided by Nav2 and is called Navigate To Pose and Pause Near Goal-Obstacle [2].

This behavior tree implements replanning every second, and clears the global costmap in the event of no available path. This dynamic replanning is helpful in a multi-robot scenario, as it accounts for the other robots and prevents collisions. When near a goal, if the path is significantly longer due to an obstacle, it waits. This works well if the obstacle is another robot, which will move and clear the path.

4 Evaluation

4.1 Complexity Analysis

4.1.1 Tracking Items

Adding an item to be tracked is an $O(1)$ operation. Getting an item for a robot involves sorting the whole list based on distance between the item and the robot. This results in a $O(n \log n)$ operation, which might be performed very regularly when a robot is in the exploring state, as the list needs to be resorted as the robot moves around. When multiple robots are spawned, they can only collect items from different segments to spread them around the map, so this could be used to prevent sorting irrelevant items by maintaining an item list for each segment. Alternatively, a k-d tree could be used to more efficiently find the closest item to the robot, which could be done in average $O(\log n)$.

4.1.2 Path Planning

The Nav2 behavior tree uses a grid based approach to planning, using a global costmap. The Navfn planner is used, and uses the A* algorithm. With a larger map this can become inefficient, and memory usage will increase exponentially with resolution. However in the case of this scenario with a bounded map, it is suitable.

4.1.3 Localization

The system uses Adaptive Monte Carlo Localization (AMCL), which uses a particle filter to keep track of the robot against a known map [3]. The size of the particle set here trades off accuracy with computational efficiency.

4.2 Descriptive

The system is highly efficient in item retrieval, and the effectiveness is not lost as the number of robots increases. The location of items is determined by converting the 2D image coordinates into 3D world coordinates using the intrinsic values of the camera and reversing the perspective projection. This proves to be very powerful when combined with the item_tracker, meaning robots being in the Exploring state is extremely rare. This method does result in the tracking of ghost items, but the system quickly recovers and is able to route the robot to a nearby item. The benefit of being able to track items between the robots outweighs the impact of navigating to a ghost item completely. This system also enables the usage of Nav2 for all navigation, which enables more deliberative actions and collision avoidance. The system could however benefit from a more low-level collision avoidance solution, such as implementing a collision monitor.

4.3 Data Analysis

To analyse the impact of the number of robots, the amount of items collected in a minute was recorded for a singular run with 1, 2, and 3 robots.

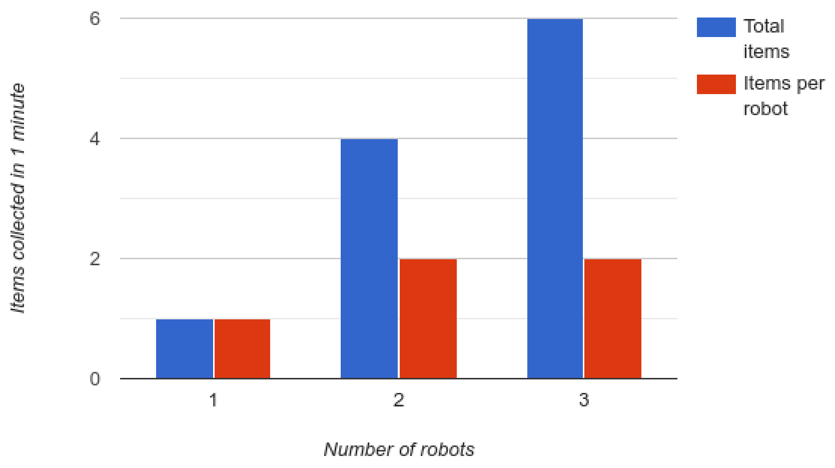

Figure 4: A graph of items collected in a minute against the number of robots

Interestingly, the singular robot had the lowest performance, after which the items per robot remained the same. This implies that the benefit from sharing item position data also outweighs the impact of having more robots and having to avoid collisions.

5 Safety and Ethics

The robotic system would pose safety risks if operating in the same environment as humans. The collision avoidance would have to be more robust and implemented at a lower-level to be deployed in a real-world scenario.

The ability for an operator to oversee or handover a robotics system could be developed further, with RViz being used for monitoring, and teleop packages used for handover, and potentially implementing emergency stop mechanisms.

The use of camera data should not pose a concern, as it is not stored, and the data is not used in a way to violate an individual's privacy. However, if the system was to be deployed at scale, it may face public backlash due to the camera collecting and processing data (even though it is not stored).

6 Simulation Scenario

The overall strategy implemented works well, and the impact of explicit communication in tracking items greatly increases performance. The strategy makes a number of assumptions, namely that all 4 zones exist and that their location is known. It also assumes that the goal is to maximise the number of items collecting, and disregard the value associated with each item. The system can be run in 3 scenarios: with 1, 2, or 3 robots. While these are not the most diverse scenarios, they enable the system to be deployed no matter how many Turtlebots are available, which could be a real obstacle in deploying this system for some users.

The different scenarios are accounted for by assigning each robot a different segment. In the 1 robot scenario, the robot covers the whole area. In the 2 robot scenario, each robot only collects items on either the left or right side of the area. Zone returning is not restricted to enable colours to be correctly returned. In the 3 robot scenario, one robot takes the whole bottom half, which has a greater line of sight with less obstacles, and the other two robots split the top half vertically. By making these assumptions, the system is able to perform extremely well, averaging an object returned every 10 seconds when running with 3 robots.

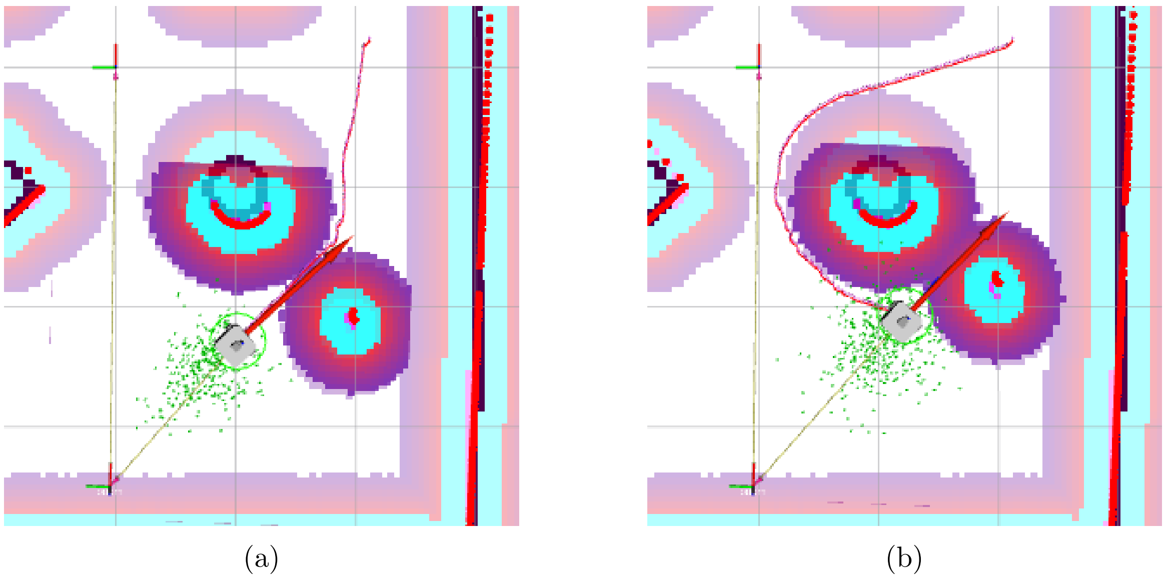
(a) (b)

Figure 5: A robot replans its route after being blocked by another robot

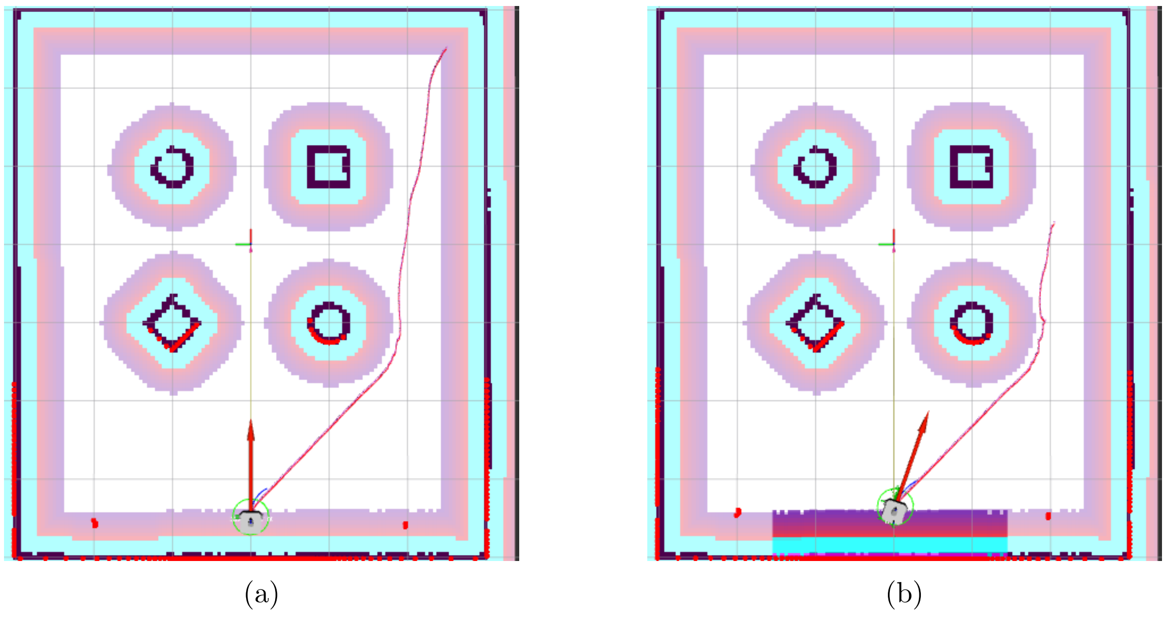
(a) (b)

Figure 6: A robot begins tracking an item observed by another robot

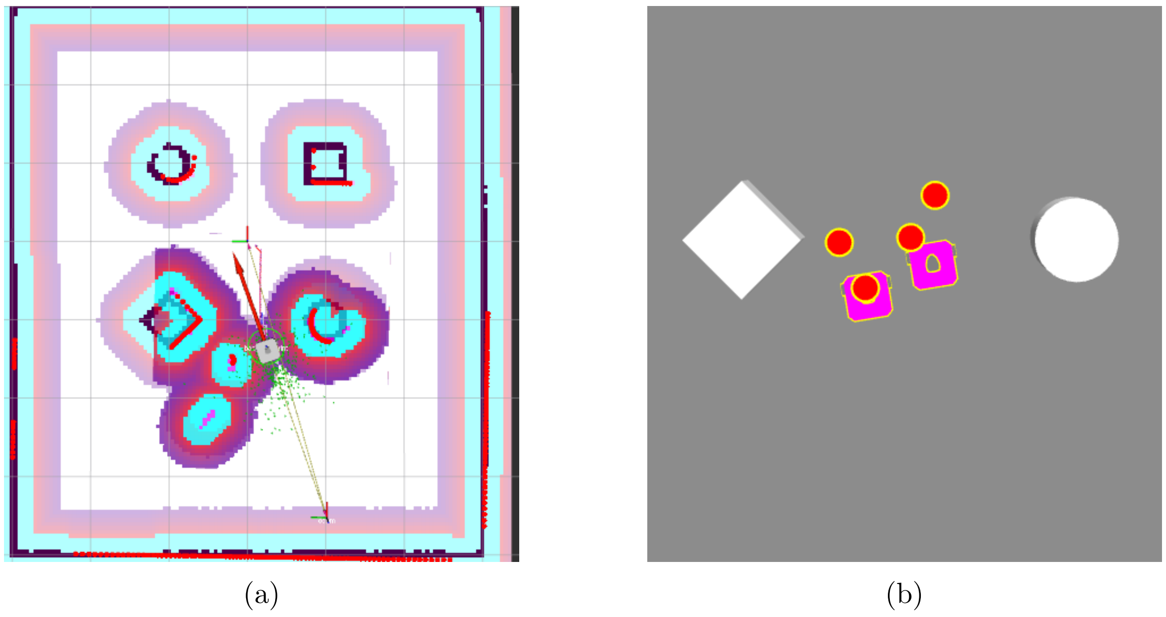

Figure 7: Close navigation between 2 robots

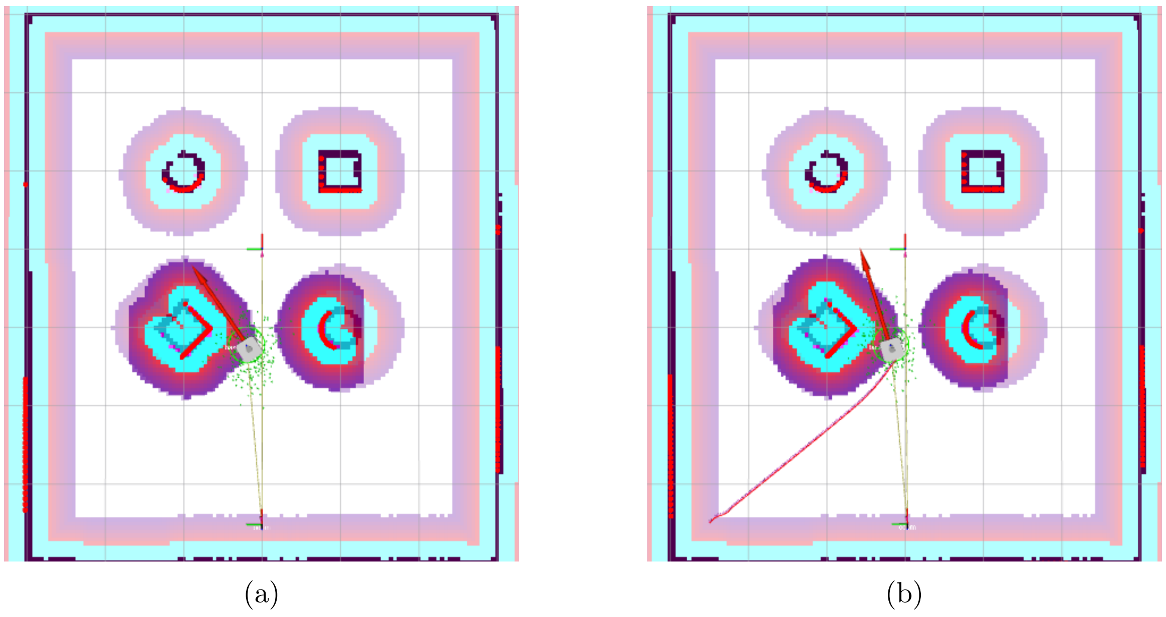

Figure 8: A robot is directed to the closest zone after picking up an item

References

[1] Manuel Schrick et al. “A microservice based control architecture for mobile robots in safety-critical applications”. In: *Robotics and Autonomous Systems* 183 (2025), p. 104795. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2024.104795. URL: https://www.sciencedirect.com/science/article/pii/S0921889024001799.

[2] *Behavior Trees: Navigate to Pose and Pause Near Goal Obstacle*. https://docs.nav2.org/behavior_trees/trees/nav_to_pose_and_pause_near_goal_obstacle.html. Accessed: 2025-01-12.

[3] *AMCL (Adaptive Monte Carlo Localization)*. http://wiki.ros.org/amcl. Accessed: 2025-01-12.