Restaurant Automation – Documentation

1. Overview

This project implements an **autonomous restaurant service robot** using **ROS 2 Humble**, **Nav2**, and **Gazebo**.

The system consists of:

- **butler_bot** → Defines the custom robot (URDF, sensors, services).
- **restaurant** → Defines the restaurant simulation environment (world + map).
- **butler_order_manager** → Manages orders and robot tasks using a **Behavior Tree (BT)** with cancel-handling logic.

The robot:

1. Waits for a customer order.
2. Navigates to the kitchen.
3. Waits for food loading.
4. Navigates to the customer's table.
5. Waits for food unloading.
6. Returns to home position.
7. Handles **order cancel** or **timeout failures** gracefully.

2. Package Details

□ butler_bot

- **URDF**: Defines robot geometry (base, wheels, sensors).
- **Services**:
  - move_to_home → Robot navigates to predefined home pose.
  - move_to_pose → Robot navigates to target pose.
- **Launches**:
  - Bringup launch files for simulation.

□ restaurant

- **World**: Restaurant layout in **SDF** (kitchen, tables, pathways).
- **Map**: Generated map for Nav2.

□butler_bot_interfaces

This package defines all custom service and action files used by the framework for order

handling and robot execution. Currently this package is missing, but equivalent standard messages were used to test the system.

□ butler_order_manager

This package manages the **orders, dispatch, and execution logic**.

Node 1 – Order Server (Service Server: Order.srv)

- **Role**: Acts as the entry point for orders/cancellations from the client (UI/terminal).
- **Inputs**:
  - new order (e.g., *table_1, table_2, table_3*)
  - cancel order (cancel by order ID).
- **Process**:
  - Maintains a **JSON stack of orders** (active, completed, canceled).
  - Every update (new/cancel) immediately updates this stack.
  - Notifies **Order Executor** about current active/canceled orders.

Node 2 – Order Dispatcher (Action Client)

- **Role**: Continuously monitors the order stack.
- **Process**:
  - Picks the **top-most valid order**.
  - Sends it as a **goal** to the **Order Executor (Action Server)**.
  - Waits for result (success or cancel).
  - Once complete, moves to the next order.

Node 3 – Order Executor (Action Server)

- **Role**: Responsible for **executing delivery workflow** for each order.
- **Main functions**:
  1. **Movement control** → Calls robot services from butler_bot:
     - /move_to_home
     - /move_to_pose (kitchen, table_1, table_2, table_3).
  2. **Load Machine Sensor check** → Subscribes to sensor data:
     - 0 = Loaded, 1 = Unloaded.
     - Robot waits at kitchen until tray = loaded.
     - Robot waits at table until tray = unloaded.
  3. **Cancellation handling**:
     - If order is **canceled while executing**:

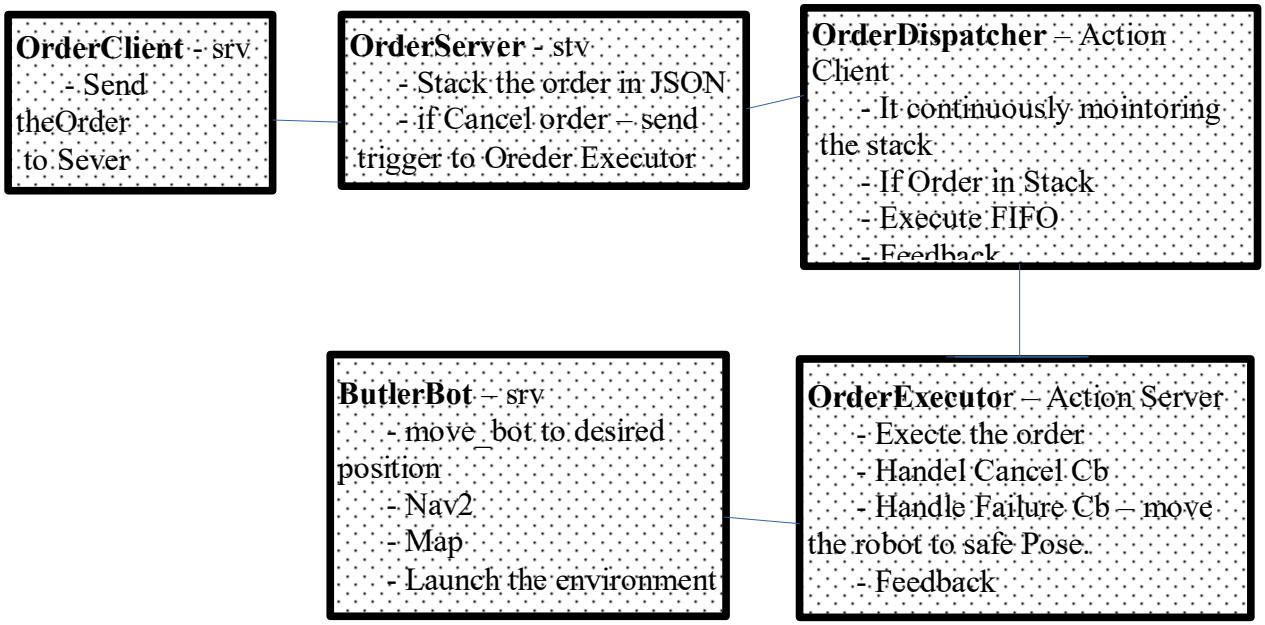
- If robot has already picked up food → return food to kitchen → go home.
- If robot hasn't picked food yet → directly go home.

- If tray not loaded/unloaded in timeout (20 sec) → cancel order → go home.

4. **Feedback & Result**: Sends progress updates to dispatcher + final result.

-

3. Code Workflow Diagram:



4. Workflow – Order Execution & Cancel Handling

Normal Order Flow

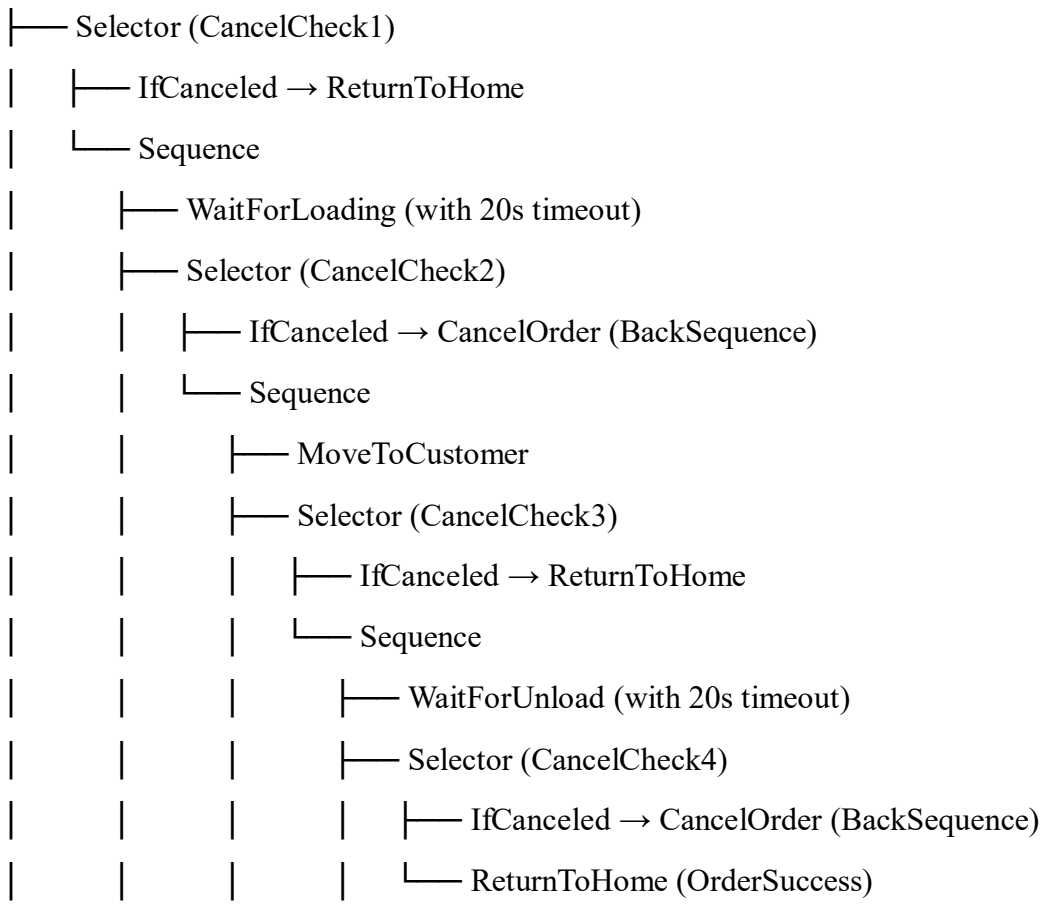
1. **Order Received**
2. Move to **Kitchen**
3. Wait for **Food Load (20s)**
4. Move to **Customer Table**
5. Wait for **Unload (20s)**
6. Return to **Home**

Root (BT - Structured)

└── Sequence

  ├── ReceiveOrder

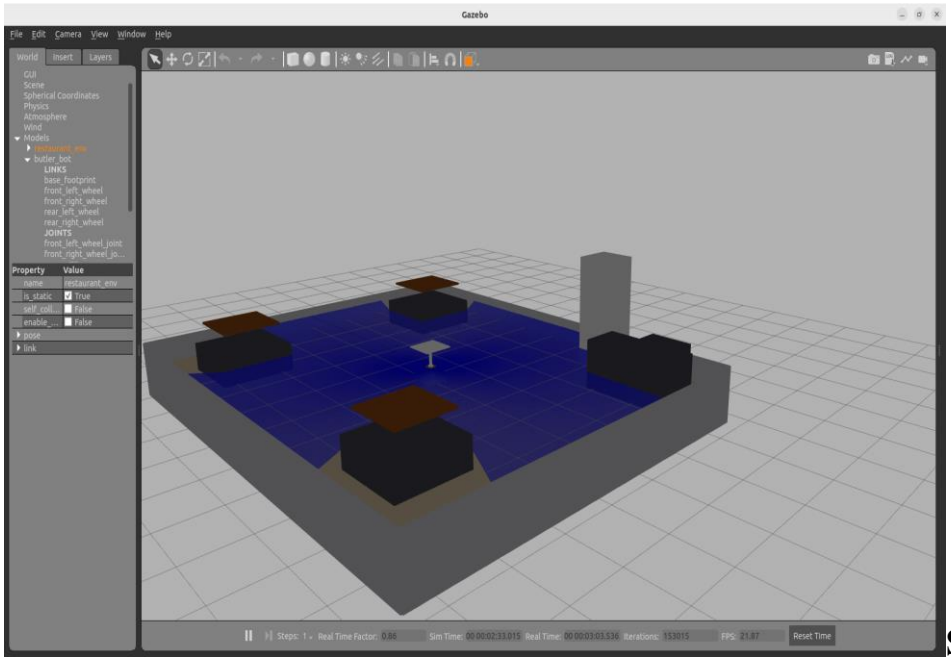
  ├── MoveToKitchen



7. How to Run

Launch Restaurant + Bot

ros2 launch butler_bot butler_restaurant_env.launch.py

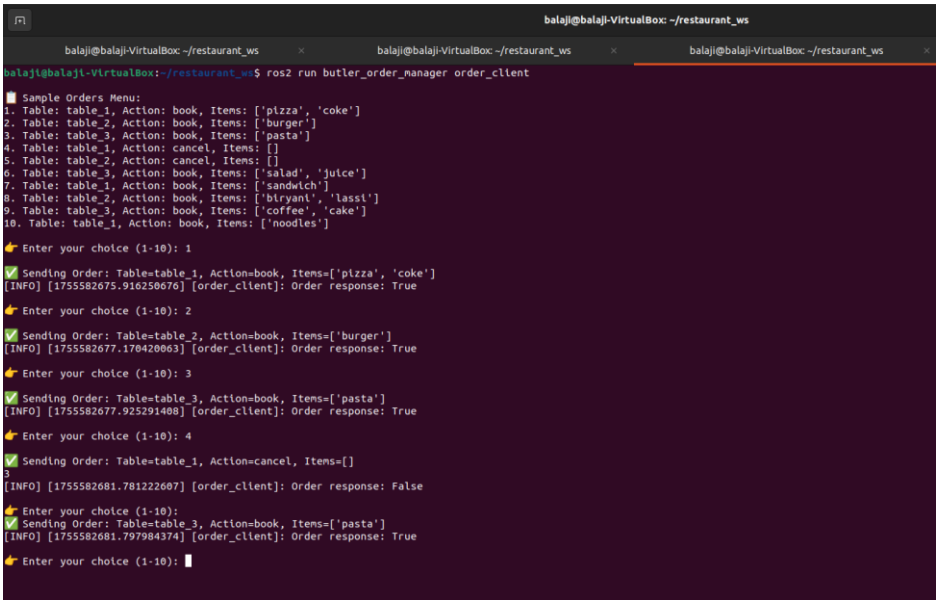


Start Order Manager

ros2 launch butler_order_manager butler_order_manager.launch.py

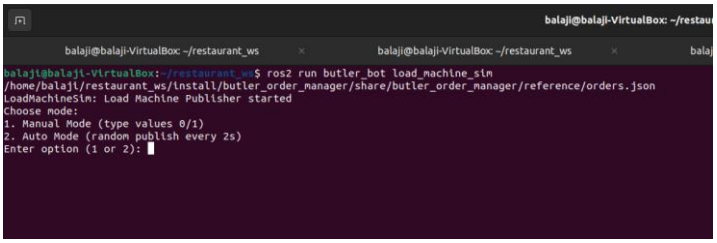
Order/Cancel:

ros2 run butler_order_manager order_client



Load Machine Sensor Simulation

ros2 run butler_bot load_machine_sim



8. Completetion Summary:

Functionalities Implemented :

Robot movement: Achieved using Navigation Stack.

- **Order Manager Stack**: Full pub/sub, services, actions flow
- **Test Cases Covered (7 total)**:

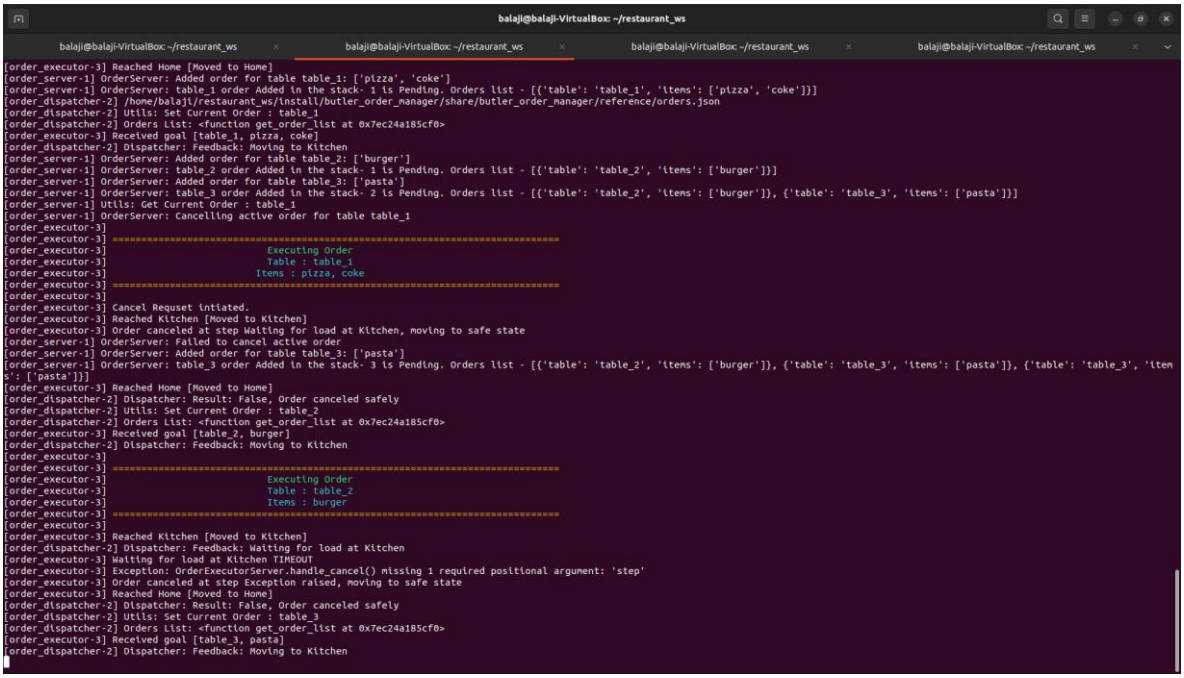
1. Place order → deliver → return home
2. Cancel before pickup → abort & return home
3. Cancel after pickup → return food → go home
4. Timeout waiting for tray load (20s) → cancel → return home
5. Multiple orders → stack management
6. Load/unload tray correctly handled
7. Success flow → dispatcher moves to next order

- **Custom Robot Model**: Mobile base + controllers configured
- **Restaurant Environment**: Kitchen + tables + map created
- **Order feedback loop** (Executor → Dispatcher → Client)

Conclusion

Even without the final map integration, the project demonstrates:

- A **working restaurant robot automation framework** in ROS 2
- Full **order workflow with cancellations, timeouts, and recovery actions**
- Robot can **move, accept orders, and manage tasks** in simulation
- Provides a solid base to add navigation on top of the existing framework
- Reference image i attached below.


balaji@balaji-VIrtualBox: ~/restaurant_ws
balaji@balaji-VirtualBox: ~/restaurant_ws
balaji@balaji-VirtualBox: ~/restaurant_ws
balaji@balaji-VirtualBox: ~/restaurant_ws
balaji@balaji-VirtualBox: ~/restaurant_ws
[order_executor-3] Reached Home [Moved to Home]
[order_server-1] OrderServer: Added order for table table_1: ['pizza', 'coke']
[order_server-1] OrderServer: table_1 order Added in the stack- 1 is Pending. Orders list - [{'table': 'table_1', 'items': ['pizza', 'coke']}]
[order_dispatcher-2] /home/balaji/restaurant_ws/install/butler_order_manager/share/butler_order_manager/reference/orders.json
[order_dispatcher-2] Utils: Set Current Order : table_1
[order_dispatcher-2] Orders List: <function get_order_list at 0x7ec24a185cf0>
[order_executor-3] Received goal [table_1, pizza, coke]
[order_dispatcher-2] Dispatcher: Feedback: Moving to Kitchen
[order_server-1] OrderServer: Added order for table table_2: ['burger']
[order_server-1] OrderServer: table_2 order Added in the stack- 1 is Pending. Orders list - [{'table': 'table_2', 'items': ['burger']}]
[order_server-1] OrderServer: Added order for table table_3: ['pasta']
[order_server-1] OrderServer: table_3 order Added in the stack- 2 is Pending. Orders list - [{'table': 'table_2', 'items': ['burger']}, {'table': 'table_3', 'items': ['pasta']}]
[order_server-1] Utils: Get Current Order : table_1
[order_server-1] OrderServer: Cancelling active order for table table_1
[order_executor-3]
[order_executor-3] ==
[order_executor-3] Executing Order
[order_executor-3] Table : table_1
[order_executor-3] Items : pizza, coke
[order_executor-3] ==
[order_executor-3]
[order_executor-3] Cancel Requset intiated.
[order_executor-3] Reached Kitchen [Moved to Kitchen]
[order_executor-3] Order canceled at step Waiting for load at Kitchen, moving to safe state
[order_server-1] OrderServer: Failed to cancel active order
[order_server-1] OrderServer: Added order for table table_3: ['pasta']
[order_server-1] OrderServer: table_3 order Added in the stack- 3 is Pending. Orders list - [{'table': 'table_2', 'items': ['burger']}, {'table': 'table_3', 'items': ['pasta']}, {'table': 'table_3', 'items': ['pasta']}]
[order_executor-3] Reached Home [Moved to Home]
[order_dispatcher-2] Dispatcher: Result: False, Order canceled safely
[order_dispatcher-2] Utils: Set Current Order : table_2
[order_dispatcher-2] Orders List: <function get_order_list at 0x7ec24a185cf0>
[order_executor-3] Received goal [table_2, burger]
[order_dispatcher-2] Dispatcher: Feedback: Moving to Kitchen
[order_executor-3]
[order_executor-3] ==
[order_executor-3] Executing Order
[order_executor-3] Table : table_2
[order_executor-3] Items : burger
[order_executor-3] ==
[order_executor-3]
[order_executor-3] Reached Kitchen [Moved to Kitchen]
[order_dispatcher-2] Dispatcher: Feedback: Waiting for load at Kitchen
[order_executor-3] Waiting for load at Kitchen TIMEOUT
[order_executor-3] Exception: OrderExecutorServer.handle_cancel() missing 1 required positional argument: 'step'
[order_executor-3] Order canceled at step Exception raised, moving to safe state
[order_executor-3] Reached Home [Moved to Home]
[order_dispatcher-2] Dispatcher: Result: False, Order canceled safely
[order_dispatcher-2] Utils: Set Current Order : table_3
[order_dispatcher-2] Orders List: <function get_order_list at 0x7ec24a185cf0>
[order_executor-3] Received goal [table_3, pasta]
[order_dispatcher-2] Dispatcher: Feedback: Moving to Kitchen