BUTLER_BOT

Author : Yohan K

Mail : 126179055@sastra.ac.in

There are 3 nodes for the working of the butler bot.

1. order_manager.py

2. confirmation.py

3. robot_controller.py

order_manager.py

Usage:

-> cd ~/butlerbot_ws

-> source the setup file

-> ros2 run butler_bot order_manager

Example input:

-> Single order

> a1

a stands for add, 1 is the table and order number.

This adds and publishes order [1] to order_queue topic.

-> Group order

> a1 a2 a3

This adds and publishes order [1,2,3] to order_queue topic.

confirmation.py

Usage:

-> cd ~/butlerbot_ws

-> source the setup file

-> ros2 run butler_bot confirmation

Example input:

-> depending upon the robot_controller's needs it will ask for confirmations

-> To respond with confirm

> 1

-> To NOT confirm

> 0 or (after timeout automatically input will be 0)

Robot_controller.py

Usage:

-> cd ~/butlerbot_ws

-> source the setup file

-> ros2 run robot_controller

Example Output:

-> It will display the robots various movements inside the cafe, no dynamic input from the user is needed for this node.

Code Explanation :

1. Order Manager Node (order_manager.py)

Overview:

The OrderManager node is responsible for managing and publishing a queue of orders. This node interacts with the user to add or remove orders and then publishes the updated order list to the order_queue topic.

Key Components:

- **Orders List (self.orders)**:
  - A list that holds the current orders.
- **Publisher (self.order_publisher)**:
  - Publishes the order queue to the order_queue topic as an Int32MultiArray.

Functions:

1. **add_order(self, order_number)**:
   - Adds an order to the self.orders list.
   - The order number is formatted as a string (Order <number>).
2. **publish_order_queue(self)**:
   - Converts the self.orders list into a list of integers representing order numbers.
   - Publishes this list to the order_queue topic.

Working:

- The node runs a loop where it accepts user input to add (a<number>) or remove (r<number>) orders.

- After processing the user input, it publishes the current order queue and resets the self.orders list.

2. Robot Controller Node (`robot_controller.py`)

Overview:

The `RobotController` node simulates a robot that processes orders by moving to specific locations (e.g., the kitchen and table). It interacts with the `order_queue` topic to receive orders and communicates with the `Confirmation` node to get confirmation on actions.

Key Components:

- **Subscribers (`self.subscription`)**:
  - Subscribes to the `order_queue` topic to receive the list of orders.
- **Clients (`self.kitchen_service_client`, `self.table_service_client`, `self.order_cancel`)**:
  - These clients interact with the `Confirmation` node to get confirmations from the kitchen and table, and to handle order cancellations.
- **Callback Groups (`self.callback_group_X`)**:
  - Ensures that callbacks are handled in separate threads for concurrent processing.

Functions:

1. **`order_callback(self, msg1)`**:
   - Processes incoming orders and passes them to `process_order` for further handling.
2. **`process_order(self, order_numbers)`**:
   - Simulates the robot's movement and processes each order by interacting with the `Confirmation` node.
   - Handles up to robot moving to the kitchen. Then passes the control to process_single_order() for table delivery

    - After table delivery checks for return to kitchen flag, if True returns to kitchen before going back to home state.
3. **`process_single_order(self, order_number)`**:
    - Simulates the robot delivering an order to a table and handles confirmation.
    - Sets return to kitchen flag ==1 when an order is not attended or cancelled.
4. **`simulate_movement(self, message)`**:
    - Prints a message to simulate the robot's movement.
5. **`order_cancellation(self)`**:
    - Checks with the `Confirmation` node to determine if an order should be cancelled.
6. **`request_kitchen_confirmation(self)`**:
    - Requests confirmation from the kitchen before proceeding.
7. **`request_table_confirmation(self, order_number)`**:
    - Requests confirmation from the table after the robot delivers the order.

Working:

- The node subscribes to the `order_queue` topic, and for each received order, it simulates the robot's movements to the kitchen and tables.
- It interacts with the `Confirmation` node to request kitchen and table confirmations and to check for order cancellations.

3. Confirmation Node (`confirmation.py`)

Overview:

The `Confirmation` node provides services for confirming actions related to kitchen and table orders and handles order cancellations. It acts as a server for the `RobotController` node's client requests.

Key Components:

- **Services (`self.kitchen_service`, `self.table_service`, `self.order_cancellation`)**:
    - Handles service requests from the `RobotController` node for kitchen/table confirmations and order cancellations.

Functions:

1. **`handle_kitchen_confirmation(self, request, response)`**:
    - Receives a service request to confirm an order at the kitchen.
    - Asks for user input to confirm or deny the order.
2. **`handle_table_confirmation(self, request, response)`**:
    - Similar to the kitchen confirmation but handles the confirmation at the table.
3. **`cancel_order(self, request, response)`**:
    - Receives a service request to cancel an order and prompts the user for confirmation.
4. **`get_confirmation(self, prompt, timeout)`**:
    - A helper function that prompts the user for confirmation within a specified timeout.
    - Uses Python's `signal` module to handle the timeout.

Working:

- This node waits for service requests from the `RobotController` node.
- Depending on the service request (kitchen confirmation, table confirmation, or order cancellation), it prompts the user for input and returns the result to the caller.

Identified Bugs:

-> In the robot_controller node (robot_controller.py) spinning of the executor happens only once, the robot executes only one set of order or receives order_queue data only once.

Temporary Solution :

1. Restart the robot_controller node (robot_controller.py) before giving next order to order_manager node (order_manager.py).